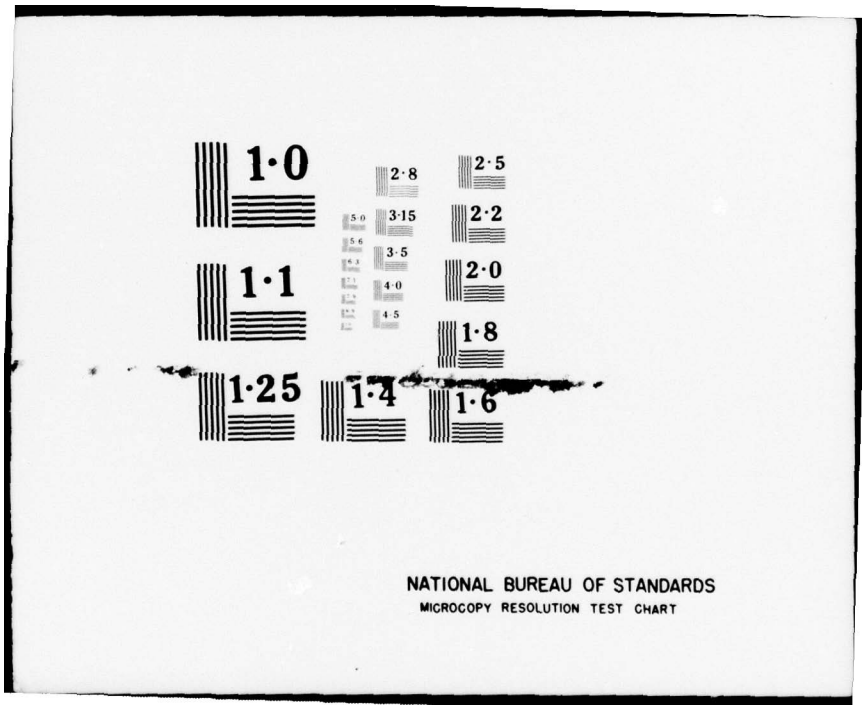
AD-A040 104 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JAVS TECHNICAL REPORT. REFERENCE MANUAL.(U)
APR 77 C GANNON, N B BROOKS F30602-76-C-0233
UNCLASSIFIED RADC-TR-77-126-VOL-2 NL

1 OF 3
ADA
040104


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

RADC-TR-77-126, Volume II (of three)
Final Technical Report
April 1977

JAVS TECHNICAL REPORT
Reference Manual

General Research Corporation

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

Some of the pages of this report are not of the highest printing quality but because of economical consideration, it was determined in the best interest of the government that they be used in this publication.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED: Frank S LaMonica

FRANK S. LA MONICA
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: RADC-TR-77-126, Volume II (of three)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*: JAVS TECHNICAL REPORT Reference Manual	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, May 76 - Nov 76
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): C. Gannon, N. B. Brooks	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0233
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Research Corporation, P. O. Box 3587, Santa Barbara CA 93105	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63728F, 55500838
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIM), Griffiss AFB NY 13441	12. REPORT DATE: April 1977
	13. NUMBER OF PAGES: 201
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Same	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer:
Frank La Monica (ISIS)

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Software Testing, Software Verification, JAVS, Automated Verification System, Computer Software.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The JOVIAL Automated Verification System (JAVS) is a tool for analyzing source programs written in the J3 dialect of the JOVIAL language. From the user's viewpoint, JAVS consists of a sequence of processing steps which (1) analyze his JOVIAL source text, (2) guide him in preparing test cases for his code, and (3) analyze the results of tests executed by his code.

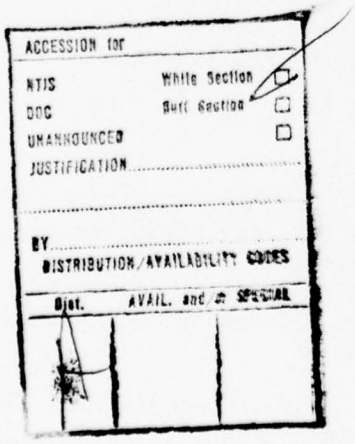
The purpose of this document is to describe in detail JAVS processing and each of the JAVS commands. The organization of the Reference Manual follows a

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

top-down approach. Starting with a system level description in the first section, each subsequent section describes a subset of the total system in detail.



LIST OF JAVS REPORTS

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements.

- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: one describing the JAVS system and the other containing the description of each JAVS command in alphabetical order. The Appendices include a complete listing of all error messages directly produced by JAVS processing.

- JAVS Technical Report: Vol. 3, Methodology Report. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described.

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter.

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed

to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module dependence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol usage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

- JAVS Preprocessor for JOVIAL. This report, prepared for GRC by its subcontractor, System Development Corporation (SDC), describes the software for the JAVS-2 component: its origin as the GEN1 part of the SAM-D ED Compiler, the modifications made in GEN1 to adapt the code for JAVS-2, the JAVS-2 code modules, and the data structures. It contains excerpts of other SDC reports on the SAM-D ED JOVIAL Compiler System. The report reflects the status of the software for JAVS-2 as delivered by SDC to GRC in September 1974. The description of JAVS-2 software contained in the System Design and Integration report reflects the status of JAVS-2 as delivered to RADC by GRC in September 1975 and thereby supercedes the SDC report.

- JAVS Final Report. The final report for the project describes the implementation and application of a methodology for systematically and comprehensively testing computing software. The methodology utilizes the structure of the software undergoing test as the basis for anlaysis by an automated verification system (AVS). The report also evaluates JAVS as a tool for software development and testing.

CONTENTS

CONTENTS (Cont.)

1 INTRODUCTION

JAVS is a system of compatible tools intended to be applied during program testing to aid in the recognition of untested program paths, to assist in the development of additional testcases appropriate to improvement of testing coverage, and to automatically document the computer program. Program validation is provided by analysis of program structures, instrumentation of the system through insertion of appropriate software probes to measure testing coverage, and comprehensive reports which pinpoint paths in the program structure that remain to be exercised. In addition, guidance is provided for the generation of testcases that will assure coverage of the untested portions. JAVS can be thought of as a partner during the testing process, supplying a wide variety of automated aids to the user in comprehensive testing activities.

This Reference Manual is intended to be used along with the JAVS User's Guide[1] and describes in detail JAVS processing and each of the available JAVS commands.

The machine-dependent features are described in the JAVS User's Guide which contains a description of the overlay version of JAVS, the JAVS macro commands, examples of job setups for RADC, and file and processing time information.

The format for this Reference Manual is as follows:

Section 1: JAVS system overview and organization, library and file description, data management, command structure, example test program, START-TERM source text identification and JAVS computation directives. The last two items deal with additions the user makes to his source code before JAVS testing begins.

Section 2: Description and diagram of each processing step.

Section 3: JAVS constraints

Section 4: System level description of JAVS commands

Section 5: Description of each JAVS command

1.1 BRIEF DESCRIPTION OF JAVS

1.1.1 Processing Steps

In the standard version, JAVS is organized as a series of independent processing steps, each devoted to a unique task, that communicate through a common data base to provide thorough software verification assistance. The overlay version of JAVS maintains the same processing tasks, but it allows the user to perform multiple processing steps in a single module selection or in a single activity. The various steps and their basic functions are listed below:

BASIC, Source Text Analysis: Source text input, lexical analysis, and initial source library creation

STRUCTURAL, Structural Analysis: Structural analysis and execution path identification; library update with structure and path information

INSTRUMENT, Module Instrumentation: Program instrumentation for path coverage analysis and program performance directed by user; library update with probe test instrumentation

ASSIST, Module Testing Assistance and Segment Analysis: Testing assistance for improved program coverage

DEPENDENCE, Retesting Guidance and Analysis: Retesting requirements analysis for changed modules

Test Execution: Execution of instrumented code and analysis of directed program performance

ANALYZER, Test Effectiveness Measurement: Detailed analysis of program path coverage; execution traces and summary statistics

These steps need not be performed in the above order. Other orders may be preferable at times.

An overview of how JAVS is used in the testing process is shown in Fig. 1.1.

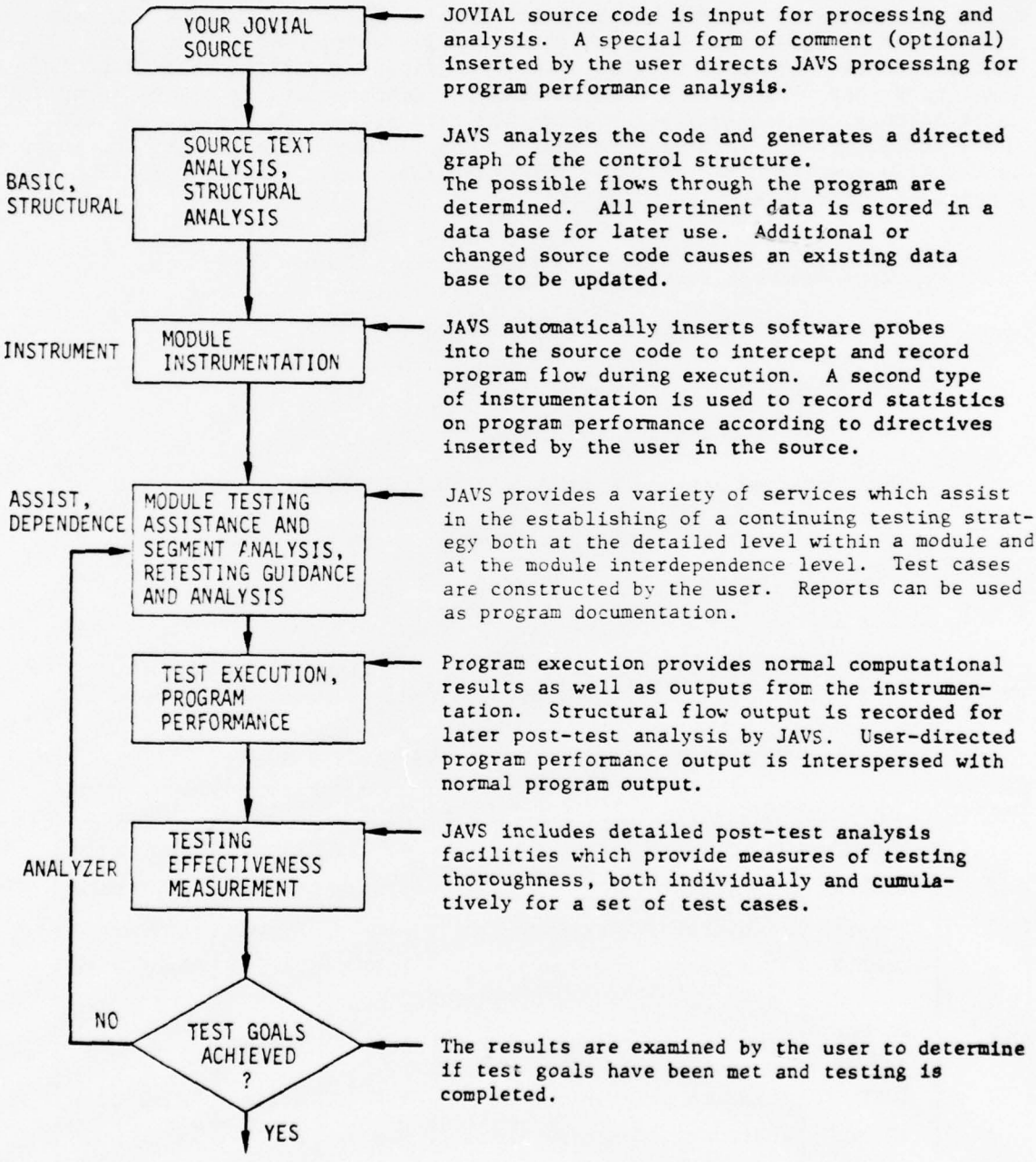


Figure 1.1. An Overview of the JAVS System

1.1.2 Library Definition and Contents

JAVS utilizes a random-access file which includes the original source text, as well as subsidiary tables that contain structural information, symbols and their classification, and other module-descriptive information. The initial random library is created during the first processing step (BASIC). The second step (STRUCTURAL) uses the initial library data to create an updated library containing program structure data as well as the source text. INSTRUMENTATION adds instrumented text to the library. The library and instrumented file are then used as input for other analyses. The major sections of a completed library are listed below:

1. Analyzed source text (BASIC)
2. Module descriptions (BASIC)
3. Statement descriptions (BASIC)
4. Symbol descriptions (BASIC)
5. Program structure information (STRUCTURAL)
6. Probe text (INSTRUMENTATION)

1.1.3 File Identification

JAVS makes use of various files during execution of the standard and overlay processing steps. The files used in JAVS processing are shown in Table 1.1.

TABLE 1.1 FILES USED IN JAVS PROCESSING

Number	Name	Function	Mode	Type	Usage
1	LIBOLD	Old library	Binary	Random	Read/write
2	LIBNEW	New library	Binary	Random	Read/write
3	LIBWSP	(Temporary) workspace	Binary	Random	Read/write
4	COMAUX	(Temporary) command iteration	BCD	Serial	Read/write
5	COMMAN COMMAC*	JAVS command input	BCD	Serial	Read only
6	LOUT	Standard printer output	BCD	Serial	Write only
7	LPUNCH	Instrumented source text	BCD	Serial	Write only
8	AUDIT	Test execution probe data	BCD	Serial	Read/write
9	READER	Source text input	BCD	Serial	Read only
10	COMMAN*	(Temporary) JAVS command workspace	BCD	Serial	Read/write

* Used only in JAVS overlay version.

1.1.4 System Execution Control

JAVS utilizes a command language to provide control over many of the processing steps. The command language provides the means to select various processing options within the selected step and, in addition, allows the user to repeatedly execute certain sequences of commands for sets of program modules contained within the library file. Certain commands are available for use in all steps and are therefore termed "universal commands." In the standard version of JAVS, some commands are unique to each processing step and are recognized only when employed within that step. The overlay version of JAVS allows commands from any processing step to be utilized in a single run.

1.1.5 Data Management

Included within the standard processing steps is a data management component which provides the following capabilities:

1. Library merging
2. Processing initialization commands
3. Module selection
4. Standard printouts of library contents

1.2 COMMAND STRUCTURE

A JAVS run is defined by a set of commands. These commands--one to a card--are freeform; blanks are ignored. The card scan ends with a period (.) or with the end of the card. Each command consists of a sequence of terms separated by a comma or an equals sign. A term with an initial digit is assumed to be an integer. If a command must be continued on more than one card, a comma must appear as the last non-blank character of each card preceding a continuation card. Up to three continuation cards may be used.

A JAVS command set always begins with some library commands ending with the command START*. Next comes a set of action and information commands describing what processing is to be performed, listing which modules are to be used, and which options are to be included. A JAVS run terminates on the END. command, which provides for correctly closing any files and writing a wrapup summary.

The commands shown in Fig. 1.2 for a hypothetical JAVS run were designed to help show the structure and use of JAVS commands.

COMMAND	TYPE
OLD LIBRARY = OLD1.	Library command
CREATE LIBRARY = NEW2.	Library command
START.	Startup command
FOR LIBRARY.	Module selection command
STRUCTURAL.	Execution command
END FOR.	Module selection command
MODULE = EXMPL1.	Module selection command
PRINT,MODULE.	Universal print command
END.	Run termination command

Figure 1.2. Hypothetical Command Sequence

*The JAVS macro commands contain the START and additional commands. The macro commands require the JAVS overlay version and are described in the JAVS User's Guide, along with other system dependent information.

1.3 EXAMPLE PROGRAM

Throughout this Reference Manual, modules from a sample JOVIAL program are used in the examples. Figure 1.3 shows a complete listing of the sample program with its five modules: COMPOOL EXCOMPL, PROGRAM EXPROGM, PROC EXAMPL1, CLOSE EXAMPL2 and CLOSE EXAMPL3. These modules are so simple that they do not in any way demonstrate the power of JAVS; their purpose is solely to provide simple output examples for each of the commands. The output from the execution of the program is shown in Fig. 1.4. This output is generated from the MONITOR statements which appear in the main program EXPROGM.

DEPENDENCE processing deals primarily with inter-module dependencies. A program with a large number of modules is used as an example to illustrate the power of inter-module analysis performed by JAVS. STRUCTURAL analysis (Sec. 2.2) is illustrated using an example procedure from the JAVS User's Guide. This small module is used to demonstrate the identification of DD-paths.

```
##.JAVSTEXT EXCOMPL PRESET##
START$
     ##COMPOOL EXAMPLE CONTAINING PRESET OUTPUT MESSAGES  ##
     COMMON MESAGE$
     BEGIN
          ITEM MSG1 H 18 P 18H(1JAVS TEST CASE    )$
          ITEM MSG2 H 18 P 18H(0RESULT LT 4       )$
          ITEM MSG3 H 18 P 18H(0RESULT EQ 4       )$
          ITEM MSG4 H 18 P 18H(0RESULT GT 4       )$
     ITEM MESSAG H 18$
     END
 TERM$


##.JAVSTEXT EXPROGM COMPUTE (EXCOMPL)##
 START$
     ##JOVIAL SIMPLE TEST PROGRAM ##
     DEFINE INTG ##I 24 S ##$
     DEFINE HLL ## H 4 ##$
     DEFINE NBYTWD ## 4 ##$
     ITEM ID HLL$
     ITEM ITER1 INTG$
     ITEM ITER2 INTG$
     ITEM ITER1A HLL$
     ITEM ITER2A HLL$
     OVERLAY ITER1 = ITER1A$
     OVERLAY ITER2 = ITER2A$
     ITEM CARD H 80$
     FILE READER H 0 R 84 V(OK) V(EOF) TAPE5$
     FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6$
     MONITOR ID, ITER1A, ITER2A$
     MESSAG = MSG1$
     OUTPUT PRINTR MESSAG$
 BG. INPUT READER CARD$
     IF READER NQ V(EOF)$
```

Figure 1.3. Example Program Used in the Reference Manual

(Continued on next page)

```
     BEGIN
          BYTE($0,NBYTWD$) (ID) = BYTE($0,NBYTWD$) (CARD)$
          BYTE($0,NBYTWD$) (ITER1A)= BYTE($9,NBYTWD$) (CARD)$
          BYTE($0,NBYTWD$) (ITER2A)= BYTE($19,NBYTWD$) (CARD)$
          EXMPL1(ITER1,ITER2)$
          CLOSE EXMPL2$      ** MAIN CLOSE **
          BEGIN
               ITER1 = 1$
               ITER2 = 1$
          END **EXMPL2**
          IF ITER1 GQ 100$
               GOTO EXMPL2$
     GOTO BG$
     END **IF**
     STOP$
PROC EXMPL1 (LIMIT1,LIMIT2)$
     BEGIN
     ITEM LIMIT1 INTG$
     ITEM LIMIT2 INTG$
     ARRAY FILL 100 INTG$
     ITEM RESULT INTG$
     ITEM INDXS INTG$
     **,TRACE,RESULT**
     IF LIMIT1 GQ 100$
     LIMIT1 = 99$
     RESULT = 4$

     FOR I = 1,1,LIMIT1$
     BEGIN
     **,EXPECT, RESULT=1,5**
          FILL ($I-1$) = I$
          RESULT = (RESULT+I)/I$
          **,ASSERT,RESULT GR 10**
          FOR J = 1,1,LIMIT2$
          BEGIN
               CLOSE EXMPL3$     ** PROC CLOSE **
               BEGIN
                    RESULT = 2*J$
               END **CLOSE**
               IFEITH J LQ 3$
                    INDXS = J$
               ORIF 1$
                    INDXS = 4$
               END **IFEITH**
               SWITCH PICK = (LABEL1,LABEL1,LABEL1,LABEL2)$
               GOTO PICK ($INDXS-1$)$
LABEL2,        GOTO EXMPL3$
LABEL1,        IFEITH RESULT LS 4$
                    MESSAG = MSG2$
               ORIF RESULT EQ 4$
                    MESSAG = MSG3$
               ORIF 1$
                    MESSAG = MSG4$
               END **IFEITH**
               OUTPUT PRINTR MESSAG$
          END **J**
     END **I**
     **,OFFTRACE,RESULT**
     END **EXMPL1**
TERM$
```

Figure 1.3. (Continued)

```
JAVS TEST CASE
••• MONITORED HOLLERITH DATA ID          = TWO
••• MONITORED HOLLERITH DATA ITER1A      =  300
••• MONITORED HOLLERITH DATA ITER2A      =  199

RESULT GT 4
••• MONITORED HOLLERITH DATA ID          = ONE
••• MONITORED HOLLERITH DATA ITER1A      =   45
••• MONITORED HOLLERITH DATA ITER2A      =   14

RESULT GT 4
```

Figure 1.4. Output from Execution of Sample Program

1.4 TEXT IDENTIFICATION

JAVS automatically separates a START-TERM sequence of JOVIAL source which contains executable JOVIAL statements (i.e., not a COMPOOL) into modules of invokable code (e.g., a PROC, a CLOSE). After the source code has been instrumented, it is reassembled into the START-TERM sequence in preparation for compilation and Test Execution. If more than one START-TERM sequence is contained on the library, it is necessary to identify each sequence separately with a unique name. JAVS has provision for naming a START-TERM sequence and distinguishing between types of text through the use of a special form of comment called a JAVSTEXT directive:

".JAVSTEXT<name><type>[(<related-text-list>)][<description>]"

where

<name>	is the name given to the START-TERM sequence (name must be 8 or less characters, in which the first 6 characters must be unique among the JAVSTEXT names)
<type>	is COMPUTE for an executable START-TERM text (e.g., a program) or PRESET for nonexecutable text (e.g., a COMPOOL)
<related-text-list>	is an optional list of related text names separated by commas (e.g., the name of a COMPOOL text used during compilation with a program text)
<description>	is optional descriptive information

The JAVSTEXT directive should appear immediately before the START for a main program or for a PROC subprogram (external PROC). In the case of CLOSE subprogram (external CLOSE), the JAVSTEXT name must be different than the CLOSE name, and the JAVSTEXT directive should appear before the CLOSE statement.

In the example program, the JAVSTEXT directive

".JAVSTEXT EXCOMPL PRESET"

is used to inform JAVS that a COMPOOL is being processed. The JAVSTEXT directive

".JAVSTEXT EXPROGM COMPUTE (EXCOMPL)"

is used to inform JAVS that an executable START-TERM sequence which references a COMPOOL (EXCOMPL) is being processed. This directive is required, even if the COMPOOL is not to be processed, if COMPOOL defined names are referenced in the program text. Only one COMPOOL may be put in the database library for each execution of the BASIC step, when processing both COMPOOL and executable text. If a COMPOOL is to be processed by JAVS, it should be analyzed only by the BASIC step. The COMPOOL will then be available on the library for documentation printing.

1.5 JAVS COMPUTATION DIRECTIVES

JAVS contains facilities for analyzing a special form of JOVIAL comment, called the JAVS directive. There are two types of JAVS directives: computation directives and text identification directives (Sec. 1.4). The computation directives, described in this section, are intended for use in monitoring computations performed by the program.

The directives are inserted by the user in the JOVIAL source before it is processed by BASIC. They are stored as separate statements in the statement block on the library created by BASIC. Later INSTRUMENT analyzes the computation directives and produces probe text (under option MODE = FULL/DIRECTIVES) which, when added to the source text, generates execution-time output interspersed with normal program output.

There are four JAVS computation directives: ASSERT, EXPECT, TRACE, and OFFTRACE. Each directive generates JOVIAL text which becomes part of the probe text. The user is cautioned that BASIC does no checking for errors in JAVS directives. Since the probe text is ultimately compiled, it must be error-free.

1.5.1 ASSERT Directive

The ASSERT directive provides the user with a means to monitor the truth value of a particular logical condition. The format of the ASSERT directive is:

```
".ASSERT,<JOVIAL-boolean-expression>"
```

The ASSERT directive may be placed wherever it is legal to place a simple IF statement followed by an assignment statement in the executable text. During Test Execution of a directive-instrumented module, a message is generated if the assertion is false.

During INSTRUMENT the ASSERT directive will generate the following code:

```
ITEM JAVS'ASSERT H 60 $

MONITOR JAVS'ASSERT $

IF NOT <JOVIAL-boolean-expression> $

JAVS'ASSERT = xxH(<JOVIAL-boolean-expression> AT STMT.

      xxxx NOT TRUE.)$
```

where xx is determined and inserted automatically. If the <JOVIAL-boolean-expression> is longer than provided for in monitor output, it will be truncated. The statement number xxxx is computed during INSTRUMENTATION to be the statement number assigned by JAVS. The statement numbers are located in the PRINT, MODULE and PRINT, JAVSTEXT reports.

1.5.2 EXPECT Directive

The EXPECT directive provides the user with a means to monitor the value of selected variables and a method by which variables which exceed certain boundaries may be detected and reported. There are two formats for the EXPECT directive:

".EXPECT,<variable> = <low-value>,<high-value>"

".EXPECT,<variable> = <value>, TOLERANCE = <error>"

In the first form the expected range of values of the named variable is specified by <low-value>,<high-value>. In the second form the expected value together with a tolerance for error of the named variable are specified. The EXPECT directive must be placed in the body of executable text for a module. During Test Execution a message is produced if the assigned value of the named variable falls outside the declared value range.

The EXPECT directive generates a monitor statement, as follows:

for ".EXPECT,<variable> = <low-value>,<high-value>"

```
MONITOR (<variable> LS <low-value> OR <variable> GR
    <high-value>)<variable>$
```

for ".EXPECT,<variable> = <value>, TOLERANCE = <error>"

```
MONITOR (<variable> LS <value> - <error> OR <variable> GR
    <value> + <error>)<variable>$
```

1.5.3 TRACE and OFFTRACE Directives

The TRACE and OFFTRACE directives allow the user to observe the effect of computation upon the variables in the program at execution time. The TRACE directive establishes the place in the program flow where tracing is to begin for a specified variable; the OFFTRACE directive establishes the place where tracing ceases. The format for these directives are:

".TRACE,<name-1>,<name-2>,...,<name-n>"

".OFFTRACE,<name-1>,<name-2>,...,<name-n>"

The value of the named variable is monitored during Test Execution. For a particular variable, monitoring commences when the TRACE directive is encountered in the execution of the instrumented module; monitoring is terminated when the OFFTRACE directive is encountered.

The TRACE directive generates the following statements:

```
".TRACE,<name-1>,<name-2>,...,<name-n>"

ITEM FLAG'1 B$

ITEM FLAG'2 B$
     .
     .
     .

ITEM FLAG'n B$

FLAG'1 = 1$

FLAG'2 = 1$
     .
     .
     .

FLAG'n = 1$

MONITOR (FLAG'1) <name-1>$

MONITOR (FLAG'2) <name-2>$
     .
     .
     .

MONITOR (FLAG'n) <name-n>$
```

The OFFTRACE directive sets the specified flags to zero.

2 JAVS PROCESSING STEPS

JAVS operates as a system of compatible tools which communicate through a common database. The order in which the processing steps are executed depends upon the task that the user wishes to accomplish. Source text (BASIC) processing must be executed first in order to build the database library.

The following tasks can be aided by JAVS, using the processing steps in the specified order:

TASK	PROCESSING STEPS
Text reformatting	BASIC
Documenting	BASIC,STRUCTURAL,ASSIST,DEPENDENCE
Debugging using directives	BASIC,STRUCTURAL,INSTRUMENT, Test Execution
Testing	BASIC,STRUCTURAL,INSTRUMENT, Test Execution,ANALYZER
Retesting	BASIC,STRUCTURAL,INSTRUMENT, Test Execution,ANALYZER,ASSIST, DEPENDENCE

In most cases BASIC, STRUCTURAL, and INSTRUMENT processing need to be performed only once.

The overlay version of JAVS consists of the same processing steps as the standard (non-overlay) version augmented by the following features:

- Smaller core size required to operate
- Macro commands to permit shorter command streams
- Operation of multiple JAVS processing steps within a single activity
- Extensibility for a planned interactive version of JAVS

2.1 SOURCE TEXT PROCESSING

The analysis of a JOVIAL program begins with BASIC, where a LIBNEW is made with information describing the texts (see Fig. 2.1). Each START-TERM sequence in the JOVIAL source is broken down into a series of invokable modules. The information saved for each module is a module descriptor block, a statement descriptor table, a statement table, a symbol locator table, and a symbol table. The results of BASIC processing can be displayed with the standard print commands (Sec. 4, PRINT, [option]).

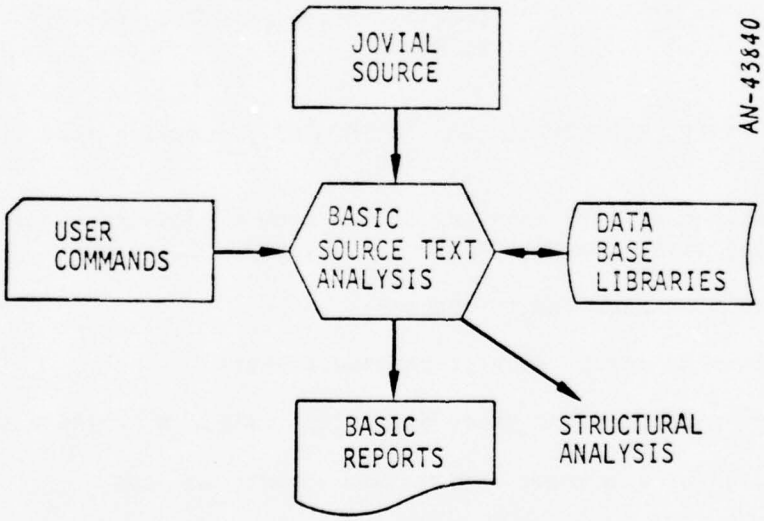


Figure 2.1. BASIC processing

2.2 STRUCTURAL ANALYSIS

The analysis of a set of modules continues with STRUCTURAL, where information describing the graphical properties of each module is added to the information generated in BASIC. STRUCTURAL begins with a library created by BASIC. In BASIC, processing is done on one or more START-TERM texts, breaking each such text down into one or more invokable modules. STRUCTURAL processing is done on a single module at a time.

Normally, the library generated during BASIC is processed as LIBNEW, a read/write library during STRUCTURAL. However, if it is used as LIBOLD, a read only library, then a LIBNEW must also be provided. If LIBOLD is used, as each module is processed by STRUCTURAL, the BASIC data for the module is first copied to LIBNEW from LIBOLD, and then the new structural data is added. STRUCTURAL processing is shown in Fig. 2.2.

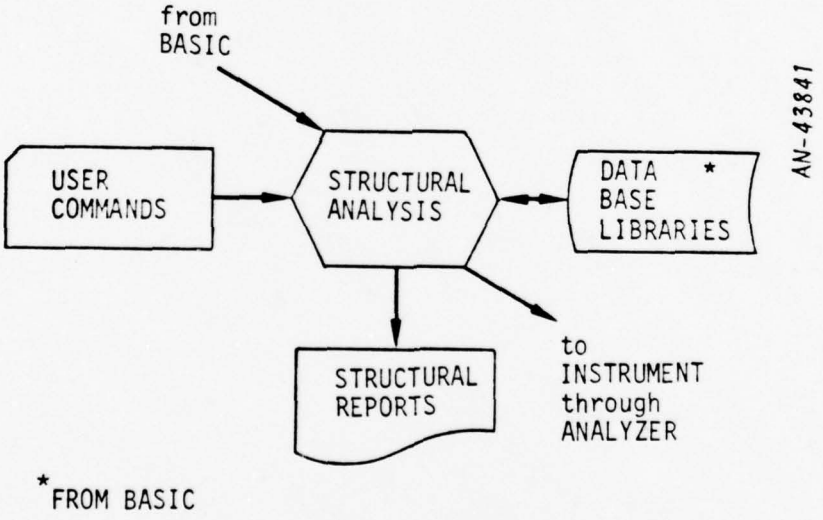


Figure 2.2. STRUCTURAL Processing

STRUCTURAL performs two types of computations:

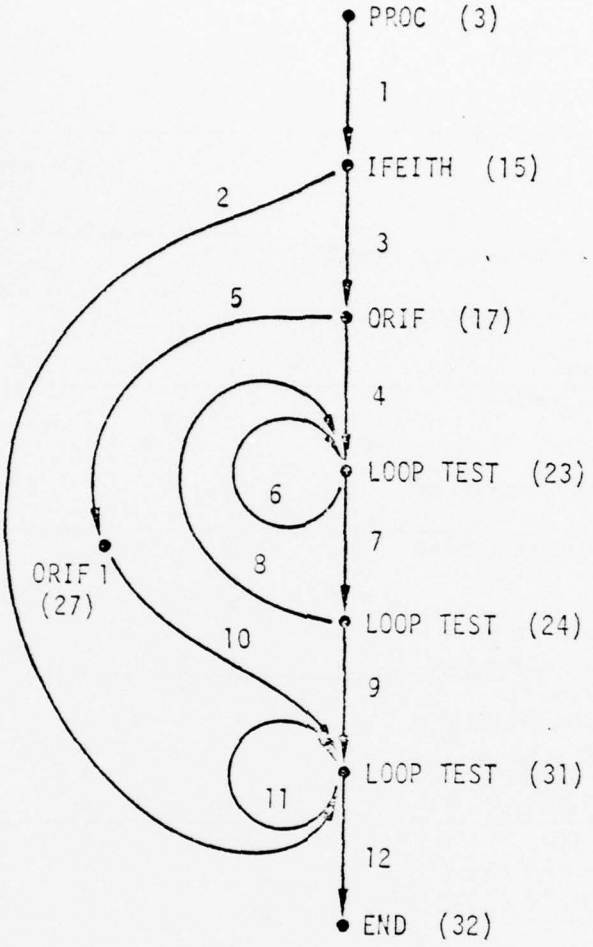
1. Inter-statement analysis which establishes statement-to-statement links (e.g., in IFEITH-ORIF statement sequences)

2. Module structural analysis which identifies the nature of program flow within the module.

The structural analysis performed by STRUCTURAL on each invokable module (program, procedure, or close) consists of determining all decision-to-decision paths (DD-paths) in the module.[2] Decision-to-decision paths are sequences of statements between decision, or branch, points. Programs are represented as a directed graph of DD-paths, as shown in Fig. 2.3 and Fig. 2.4. Figure 2.3 shows the statement membership for each DD-path in module EXAMPL. This module contains 12 DD-paths. Below each DD-path number is listed the order in which the statements are placed on each DD-path. For example, DD-path 2 consists of statements 15, 16, 29, 30, and 31, in that order.

		Statement Execution Order											
		DD-Path											
		1	2	3	4	5	6	7	8	9	10	11	12
1	'' . JAVSTEXT EXAMPL COMPUTE (COMPOL)''												
2	START												
3	PROC EXAMPL (AA = BB) $	1											
4	ITEM AA F $												
5	ITEM BB F $												
6	ARRAY CC 2 2 F $												
7	BEGIN												
8	BEGIN												
9	+10.E-001 +10.E-001 END												
10	BEGIN												
11	+10.E-001 +10.E-001 END												
12	END												
13	BEGIN	2											
14	MONITOR BB , CC $												
15	IFEITH AA LS 00.E-001 $	3	1	1									
16	BB = - AA $		2										
17	ORIF AA EQ 00.E-001 $			2	1	1							
18	BEGIN				2								
19	BB = 00.E-001 $				3								
20	FOR I = 0 , 1 , 1 $				4				2				
21	BEGIN				5				3				
22	FOR J = 0 , 1 , 1 $				6		2		4				
23	CC ($ I , J $) = 00.E-001 $				7		1,3	1	5				
24	END							2	1	1			
25	RETURN $									2			
26	END												
27	ORIF 1 $					2					1		
28	BB = AA $										2		
29	END		3								3		
30	FOR K = 0 , 1 , 1 $		4								4	2	
31	CC ($ K , 0 $) = BB / 20.E-001 $		5							3	5	1,3	1
32	END												2
33	TERM $												

Figure 2.3. Example of Structural Analysis



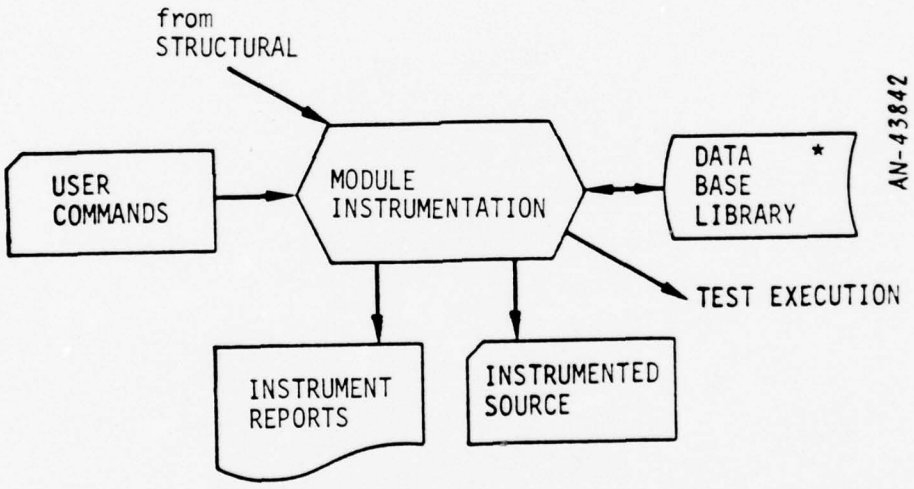
Legend

3 →	Numbers represent DD-paths
●	Dots represent decision points
IFEITH	Words represent decision statement types
(31)	Parenthetical numbers represent statement numbers

Figure 2.4. Pictoral Example of Structural Analysis

2.3 MODULE INSTRUMENTATION

This step performs the instrumentation of modules which have been processed through BASIC and STRUCTURAL. The primary result of this step is a set of probed modules which can then be compiled in instrumented form for use in Test Execution (see Sec. 2.4). The instrumented modules are logically identical to the original source code. The probe statements are saved on the library along with data generated by BASIC and STRUCTURAL (e.g., the source text, etc.). INSTRUMENT processing is shown in Fig. 2.5.



*FROM STRUCTURAL

Figure 2.5. INSTRUMENT Processing

There are two types of instrumentation generated during INSTRUMENT:

1. Structural Instrumentation. Software probes are automatically inserted into the source text at each invocation point, each return point, and each statement which begins a DD-path. Each probe includes a call to special auditing routines which capture and record information concerning flow of control in the executing module(s).

2. JAVS Directive Instrumentation. Software probes are manually inserted in the source text for JAVS directives (Sec. 1.5) which are automatically expanded into JOVIAL code to monitor the results of assignment and exchange statements during Test Execution. Each directive controls execution-time output which is interspersed with normal program output.

2.4 MODULE TESTING ASSISTANCE AND SEGMENT ANALYSIS

The purpose of ASSIST is to provide computational and output reports for use by program testers to meet the following objectives:

- To provide "source text flow" information needed to support detailed analyses of the legitimacy of test cases
- To provide for identification of properties of sequences of DD-paths

ASSIST commands are designed to provide detailed information on specific DD-paths in modules. Some of the commands require that the DD-paths for which test assistance is needed be identified. Therefore, ASSIST is usually run after an initial series of test cases is run, and additional information is needed to assist the user in setting up new test cases that give coverage of specific DD-paths. ASSIST reports are most effectively used for retesting assistance and for computer documentation in conjunction with reports from the standard print commands and from DEPENDENCE (Sec. 2.5).

ASSIST operates from LIBOLD and, under active user control, generates reports of various kinds. It is assumed that BASIC and STRUCTURAL have been completed for each module present on LIBOLD for which analyses are requested by ASSIST commands. ASSIST processing is shown in Fig. 2.6.

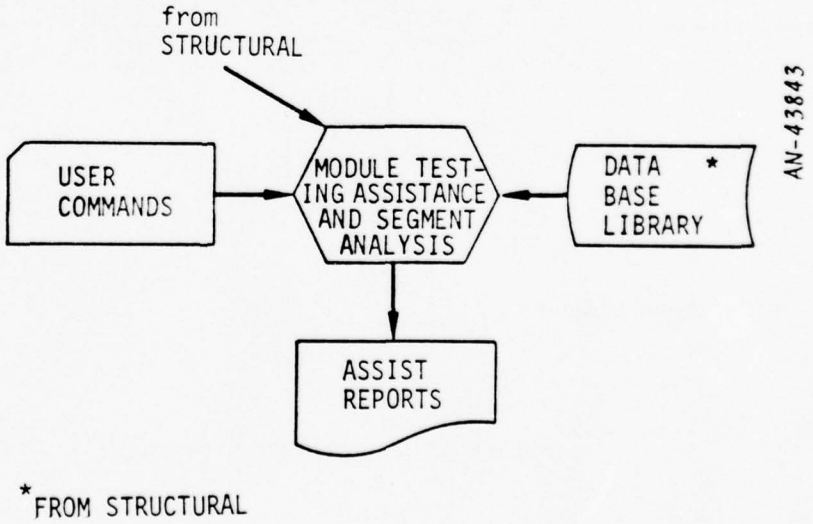


Figure 2.6. ASSIST Processing

2.5 RETESTING GUIDANCE AND ANALYSIS

The purpose of DEPENDENCE is to provide the means to assess the impact of program modifications for a set of test cases. In contrast with ASSIST (which examines the details of program flow within a module), DEPENDENCE is concerned primarily with inter-module dependencies. Taken with the reports from both ASSIST and ANALYZER, the analyses performed by DEPENDENCE allow the tester to specify the subset of test cases which must be re-executed in order to maintain comprehensive testing coverage. The reports from DEPENDENCE coupled with some of the ASSIST reports provide valuable software documentation.

DEPENDENCE operates from LIBOLD, generating reports under active user control. It is assumed that BASIC and STRUCTURAL have been completed for every module present on LIBOLD. DEPENDENCE processing is shown in Fig. 2.7.

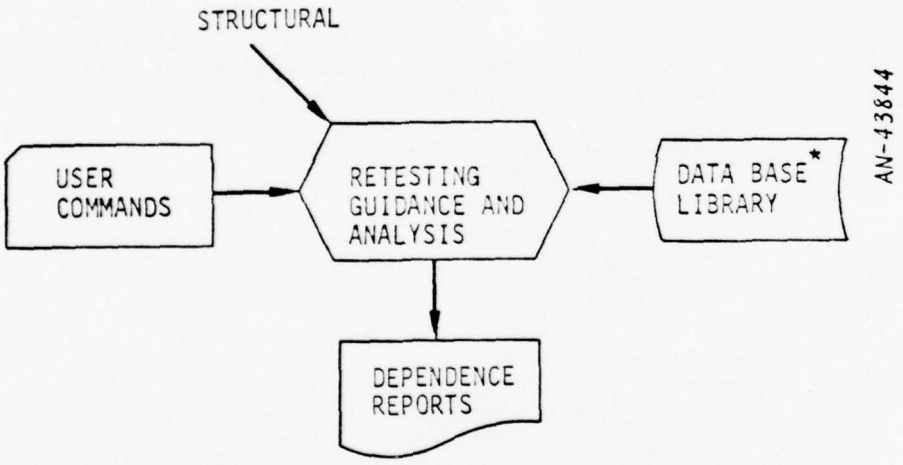


Figure 2.7. DEPENDENCE Processing

2.6 TEST EXECUTION

The purpose of Test Execution is to collect test execution data and performance data from JAVS directives while executing a set of instrumented modules produced by INSTRUMENT. Test Execution differs from other JAVS steps in that the instrumented program itself is executed, not a JAVS processing step. There is no access to the JAVS library during Test Execution. Rather, the probe data is recorded on a test file; these data include the information needed by ANALYZER to properly correlate with data on the library (e.g., module name, test name, and DD-path number).

The instrumented program under test is first compiled, then loaded with JAVS data collection modules. The program operates normally, reading its own data and writing its own outputs. During program execution, the instrumented modules call the data collection routines which record on AUDIT (the test file) an execution trace and accumulated data on module invocations and return and on DD-path traversals. Performance data resulting from JAVS computation directives (Sec. 1.5) is interspersed with normal program output for the printer. Test Execution processing is shown in Fig. 2.8.

Each Test Execution may consist of a number of test cases. The program identifies the start of each new test case by executing a call to one of the data collection routines; the end of all test cases is similarly treated. These identification calls can be manually inserted by the user in the program prior to compilation, or they can be inserted automatically during instrumentation by using the INSTRUMENT,STARTTEST or INSTRUMENT,STOPTEST commands (Sec. 5). All other instrumentation of the source is automatically performed by INSTRUMENT.

2.6.1 JAVS Data Collection for Structural Instrumentation

There are four procedures in the JAVS data collection routines which may be invoked during test execution:

- PROBE* is called at the beginning of each DD-path in an instrumented module (INSTRUMENT option DDPATHS or FULL)
- PROBM* is called upon module invocation or return in an instrumented module (INSTRUMENT option INVOCATION, DDPATHS or FULL)
- PROBI* is called to identify a new test case, or to end all test cases
- PROBD* is called to record descriptive information about a test on the test file

INSTRUMENT automatically generates the probes for PROBE and PROBM. The calls to PROBI are inserted by the user, manually or by command, in the instrumented source (see Sec. 2.6.2). The optional calls to PROBD are inserted manually by the user.

*An arbitrary name used in data collection routines.

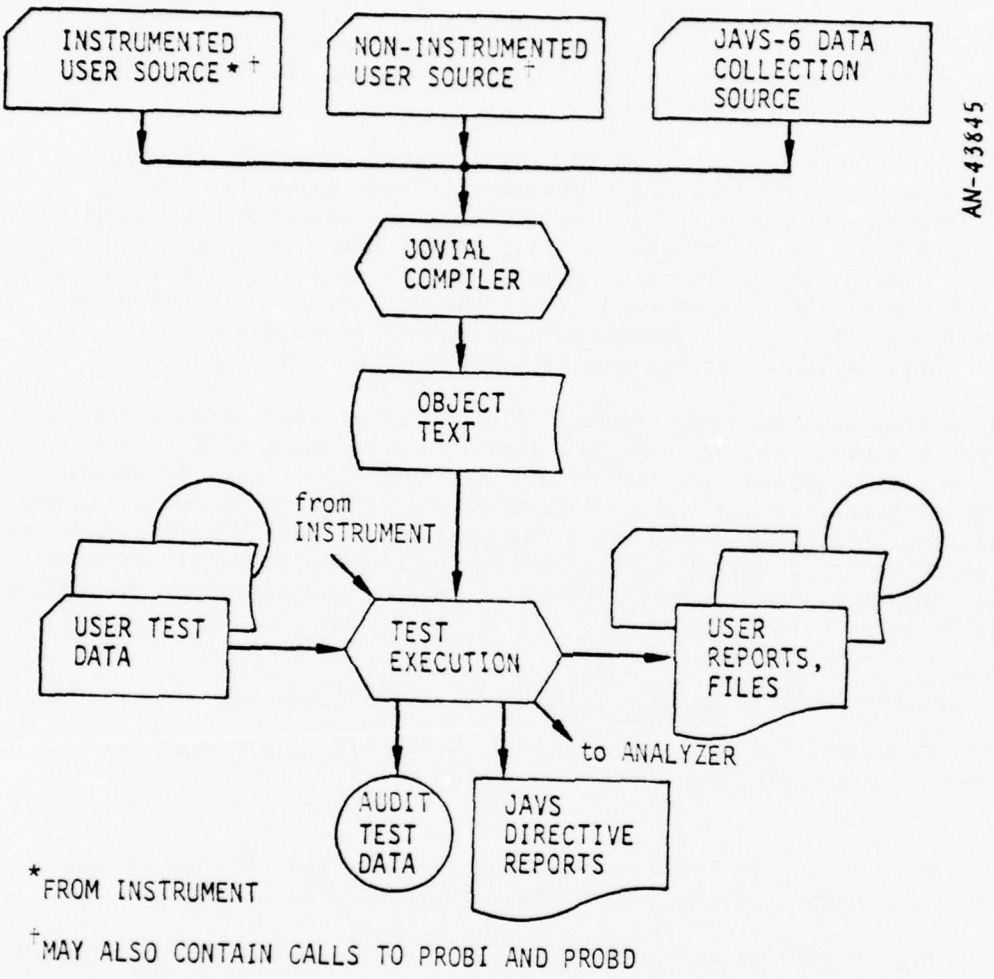


Figure 2.8. Test Execution Processing

The data collection routines utilize a COMMON block (PROBEC*) for communication with one another and for storage of data by module and DD-path. In the standard version of the COMPOOL for the data collection routines, provision is made for a maximum of 100 instrumented modules and 2000 DD-paths for those modules whose DD-paths are instrumented. If larger space is required, the data collection routines must be recompiled with appropriate changes made in their COMPOOL.

Caution: The program being analyzed may include symbols which conflict with the names used in the data collection routines for the procedures, the COMMON block, or the test file. Should such a conflict exist, (1) appropriate changes must be made in the source for the data collection routines and their COMPOOL and (2) INSTRUMENT processing must be done with the PROBE option command(s) to change the names of the data collection routines.

Summary data is accumulated in the workspace only after the first call is made to PROBI. Thereafter whenever a call is made to PROBI signalling the beginning of a new test case (or the end of all test cases) the accumulated data is recorded on AUDIT and the workspace is cleared. For each call to PROBE or PROBM, an execution trace record is recorded; for each call to PROBI, test case identification data is recorded; for each call to PROBD, descriptive information is recorded.

For each module the amount of data which it is possible to accumulate is controlled by the level of structural instrumentation done during INSTRUMENT. If the module is instrumented at the invocation level, only PROBM is called and DD-path executions cannot be recorded. If the module is instrumented at the DD-path level, both PROBM and PROBE are called, and full data acquisition can be made.

The amount of data actually recorded during a test is further controlled by a parameter to PROBI (see Sec. 2.6.2.2).

2.6.2 Compiling The Instrumented Program

During INSTRUMENT processing, the instrumented program source must first be written to a punch file from the library by using the command

PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <specification>.

while processing under JAVS execution. Next the instrumented program source is prepared for the compiler as described in the following subsections.

2.6.2.1 References to Data Collection Routines and Test File.

If the program has a COMPOOL, the procedure declarations for the data collection routines must be added to the COMPOOL.* If the program has no

*If the JOVIAL compiler accepts a list of processed COMPOOLs while compiling executable modules, then one for the source shown in Fig. 2.9 must be included in the list.

COMPOOL, the one shown in Fig. 2.9 must be provided. This is necessary in order to compile the instrumented source.

A FILE declaration for the test file (AUDIT) may also be necessary in the compilation process.* The location of the FILE declaration (i.e., in a COMPOOL or in the main program) is dependent on the JOVIAL compiler being used. Figure 2.9 contains the proper file statement for RADC.

```
**.JAVSTEXT PRCMPL PRESET = COMPOOL FOR PROBE ANALYSIS DURING EXECUTION
**
START $
        DEFINE INTG ** I 24 S ** $
        DEFINE HLL ** H 4 ** $
        DEFINE DINTG ** I 48 S ** $
        DEFINE DHLL ** H 8 ** $
PROC PROBM(MODNAM,JAVTXT,NOOPS)$
BEGIN
ITEM MODNAM DHLL $
ITEM JAVTXT DHLL $
ITEM NOOPS  INTG $
END
PROC PROBE(MODNAM,JAVTXT,OOP)$
BEGIN
ITEM MODNAM DHLL $
ITEM JAVTXT DHLL $
ITEM OOP    INTG $
END
PROC PROBI(TESNAM,TFLAG)$
BEGIN
ITEM TESNAM DHLL $
ITEM TFLAG  INTG $
END
PROC PROBD(LINE,FLAG)$
BEGIN
ITEM LINE H 80 $
ITEM FLAG   INTG $
END
COMMON PROBEF $                 **COMMON BLOCK FOR PROBE FILE**
  BEGIN
  FILE AUDIT B 84 R 0 V(OK) V(X1) V(X2) V(X3) V(EOF) 08$
  END
TERM $
```

Figure 2.9. COMPOOL for References to JAVS Data Collection Routines

* Some JOVIAL compilers require the FILE statement to be in the main program; others require it to be in a processed COMPOOL loaded with the executable code.

2.6.2.2 Test Case Identification and Test File Control.

At appropriate places in the instrumented source program (i.e., where a new test case begins and at the end of all test cases) a call to PROBI must be inserted. PROBI performs two services: it identifies each test case and controls the recording of data on the test file. PROBI has two parameters: the first is used as a test case name on ANALYZER reports and is a Hollerith name of 8 characters; the second is used to control the amount of data actually recorded on the test file.

The possible values for the test file control parameter TFLAG are shown in Table 2.1. A zero value signals the end of all test cases. A nonzero value signals the start of a new test case (and the end of the previous test case, if any). The value of the second parameter (if nonzero) determines the amount of execution tracing. If TFLAG is 1, no tracing is maintained; if TFLAG is 2, invocations and returns are traced; if TFLAG is 3, invocations, returns, and DD-paths are traced.

TABLE 2.1

TEST FILE DATA CONTROL WITH PROBI

TFLAG	SIGNAL	TEST-FILE DATA RECORDED	ANALYZER REPORT OPTIONS
0	end-of-test file	(last) test case summary	
1	new test case	test-case summary	SUMMARY, HIT, NOTHIT, MODLST, DDPATHS
2	new test case	test-case summary, module invocation/return trace	SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDPATHS
3	new test case	test-case summary, module invocation/return trace, DD-path execution trace	SUMMARY, HIT, NOTHIT, MODLST, TIME, MODTRACE, DDTRACE, DDPATHS

2.6.2.3 Test Description.

At user option, descriptive information about the test can also be recorded on the test file by inserting PROBD calls. There are two parameters to PROBD (see Fig. 2.9): the first parameter is a Hollerith test description string of 80 characters, and the second parameter controls the initialization of data workspace. The value zero indicates the start of a test set (i.e., a new set of test cases), and a nonzero value indicates the start of a new test case. PROBD calls, if used, should immediately precede PROBI calls.

Each PROBD call results in a single line of output during ANALYZER processing. There is no limitation on the number of consecutive calls to PROBD. When the test file is analyzed by ANALYZER, the descriptive information is

printed on the output file as encountered. The purpose of PROBD is to allow the user a facility for correlating program behavior to test results. For example, strategically placed PROBD invocations can identify program linkages executed for specific tests.

2.6.2.4 Compilation.

The procedure for compiling the instrumented source program with the JOVIAL compiler is as follows:

1. Punch the instrumented text after INSTRUMENT execution. (Sec. 2.6.2).

2. If the text contains structural instrumentation (INSTRUMENT default option, INVOCATION, DDPATHS or FULL) then

   - Provide COMPOOL definitions for data collection routines (Sec. 2.6.2.1).

   - Insert PROBI and PROBD calls (Secs. 2.6.2.2 and 2.6.2.3).

3. Compile the COMPOOL and the instrumented program. If the text contains JAVS directives for performance data (INSTRUMENT options DIRECTIVES, FULL), use the MONITOR option* for the JOVIAL compiler. Additional core storage may be required for compiling the instrumented code.

Figure 2.10 shows an example listing of probed text with PROBI calls inserted.

*In the present JAVS implementation, JAVS performance directives make use of the MONITOR feature available in some JOVIAL compilers, including the JOCIT JOVIAL compiler at RADC.

```
      ##. JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) ##
      START
      $
      ## JOVIAL SIMPLE TEST PROGRAM ##
      ITEM ID H 4 $
      ITEM ITER1 I 24 S $
      ITEM ITER2 I 24 S $
      ITEM ITER1A H 4 $
      ITEM ITER2A H 4 $
      OVERLAY ITER1 = ITER1A $
      OVERLAY ITER2 = ITER2A $
      ITEM CARD H 80 $
      FILE READER H 0 R H4 V(OK) V(EOF) TAPE5 $
      FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
        FILE AUDIT  R 0 R 0 V(OK) V(EOF) AUDITR $ ##TEST PROBE ##
      MONITOR ID , ITER1A , ITER2A $
          PROB1 (BH(SAMPLE   ),3)$
      PROBM     ( BH(EXPROGM ) , BH(EXPROGM ) ,     5 ) $
      PROBE     (BH(EXPROGM ), BH(EXPROGM ),     1) $
      MESSAG  =    MSG1     $
      OUTPUT PRINTR MESSAG $
   BG.
      INPUT READER CARD $
      PROB1(BH(SAMPLE   ),3)$
      IFEITH    READER NQ V(EOF) $
        BEGIN
        PROBE     (BH(EXPROGM ), BH(EXPROGM ),     2) $
          BEGIN
          BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
          BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
          BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
          EXMPL1 ( ITER1 , ITER2 ) $
          CLOSE EXMPL2 $
      ## MAIN CLOSE ##
          BEGIN
          PROBM     ( BH(EXMPL2   ) , BH(EXPROGM ) ,     1 ) $
          PROBE     (BH(EXMPL2   ), BH(EXPROGM ),     1) $
          ITER1 = 1  $
          ITER2 = 1 $
          PROBM     ( BH(EXMPL2   ) , BH(EXPROGM ) ,     0 ) $
          END
      ## EXMPL2 ##
          IFEITH    ITER1 GQ 100 $
            BEGIN
            PROBE     (BH(EXPROGM ), BH(EXPROGM ),     4) $
            GOTO EXMPL2 $
            END
          ORIF 1 $
            PROBE     (BH(EXPROGM ), BH(EXPROGM ),     5) $
          END
          GOTO     BG  $
          END
      ## IF ##
        END
      ORIF 1 $
        PROBE     (BH(EXPROGM ), BH(EXPROGM ),     3) $
      END
      PROBM     ( BH(EXPROGM ) , BH(EXPROGM ) ,     0 ) $
          PROB1 (BH(SAMPLE   ),0)$
      STOP     $
      PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $
        BEGIN
        ITEM LIMIT1 I 24 S $
        ITEM LIMIT2 I 24 S $
        ARRAY FILL 100 I 24 S $
        ITEM RESULT I 24 S $
        ITEM INDXS I 24 S $
      ##. TRACE , RESULT ##
        PROBM     ( BH(EXMPL1   ) , BH(EXPROGM ) ,    20 ) $
        PROBE     (BH(EXMPL1   ), BH(EXPROGM ),     1) $
        IFEITH    LIMIT1 GQ 100 $
          BEGIN
          PROBE     (BH(EXMPL1   ), BH(EXPROGM ),     2) $
          LIMIT1 = 99 $
          END
        ORIF 1 $
          PROBE     (BH(EXMPL1   ), BH(EXPROGM ),     3) $
        END
        RESULT  =    4    $
        FOR I    =    1    ,    1    ,    LIMIT1  $
          BEGIN
          IF     I NQ 1 $
            PROBE     (BH(EXMPL1   ), BH(EXPROGM ),    19) $
            BEGIN
```

Figure 2.10. Excerpt from Probed Text

2.6.3 Loading the Instrumented Program

If the instrumented program contains structural probes (i.e., calls to PROBM, PROBE, PROBI or PROBD) the JAVS data collection routines must be added to the load module for Test Execution. In addition, provision must be made for the test file to be saved on a suitable medium (e.g., tape or disk).

The requirements for main memory are larger in Test Execution than in normal program execution due to the following factors:

- Instrumented code is larger than non-instrumented code.
- JAVS data collection routines, workspace, and test file are needed for structural instrumentation.

Code expansion due to instrumentation depends on the nature of the source program. Experience has shown that code expansion of programs with moderately complex control structures can have 100% expansion from DD-path instrumentation. The JAVS data collection routines, although relatively small, utilize about 3000 words for data space in the standard configuration; the space required for the test file control information and buffer space is installation dependent.

2.6.4 Test Execution Input and Output

The input data requirements for test execution are those for normal program execution. There is no input data read by the JAVS data collection routines.

In addition to normal program output, two types of output result from test execution of an instrumented program: a test file (for ANALYZER) which contains execution data collected from structural instrumentation probes and printout from the JAVS performance probes. Figure 2.11 contains a sample output from JAVS performance probes.

```
JAVS TEST CASE
*** MONITORED HOLLERITH DATA ID          = ONE
*** MONITORED HOLLERITH DATA ITER1A      =  45
*** MONITORED HOLLERITH DATA ITER2A      =  14
*** MONITORED INTEGER DATA RESULT        =                    4 = 0(00000000000000000004)
*** MONITORED HOLLERITH DATA JAVS#ASSER =                        RESULT GR 10 AT STMT.   18 NOT TRUE.
*** MONITORED INTEGER DATA INDXS         =                    I = 0(00000000000000000001)
*** MONITORED HOLLERITH DATA JAVS#ASSER =                        INDXS GR 2 AT STMT.     30 NOT TRUE.
*** MONITORED HOLLERITH DATA MESSAG      = 0RESULT GT 4

RESULT GT 4
*** MONITORED HOLLERITH DATA ID          = TWO
*** MONITORED HOLLERITH DATA ITER1A      =  300
*** MONITORED HOLLERITH DATA ITER2A      =  199
*** MONITORED INTEGER DATA RESULT        =                    4 = 0(00000000000000000004)
*** MONITORED HOLLERITH DATA JAVS#ASSER =                        RESULT GR 10 AT STMT.   18 NOT TRUE.
*** MONITORED INTEGER DATA INDXS         =                    I = 0(00000000000000000001)
*** MONITORED HOLLERITH DATA JAVS#ASSER =                        INDXS GR 2 AT STMT.     30 NOT TRUE.
*** MONITORED HOLLERITH DATA MESSAG      = 0RESULT GT 4

RESULT GT 4
```

Figure 2.11. Performance Data from JAVS Directives

2.7 TEST EFFECTIVENESS MEASUREMENT

The purpose of ANALYZER is to provide test effectiveness measurement and test execution tracing in terms of module invocation, DD-paths exercised, and statements executed. For a specified set of modules, ANALYZER supplies the following measures of test case effectiveness.

- A summary at the module level of the coverage achieved.
- A table of DD-paths exercised and the number of times each DD-path was traversed.
- A list of the DD-paths not executed.
- A source listing of each module including the number of times each statement was executed.
- A source listing of the key statements for each DD-path including the number of times each DD-path was executed.

ANALYZER supplies the following types of execution tracing:

- A list of the module invocations and returns and the level of invocation, or
- A list of the DD-paths interspersed with the module invocations and returns.

In addition, ANALYZER summarizes the time spent in each module as measured by the system clock.

ANALYZER operates from LIBOLD and a saved test file AUDIT produced by Test Execution. It is assumed that BASIC through INSTRUMENT and Test Execution have been accomplished for the specified modules. ANALYZER processing is shown in Fig. 2.12.

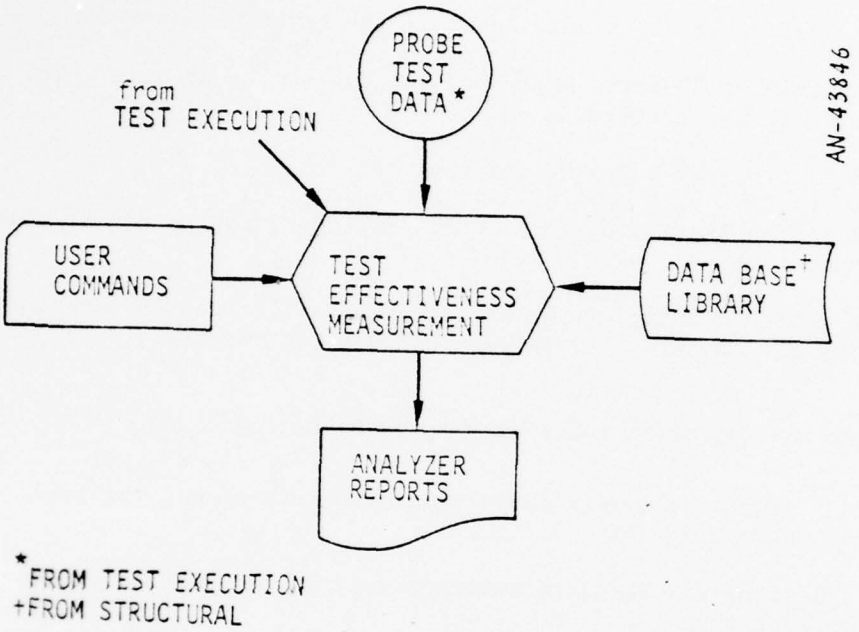
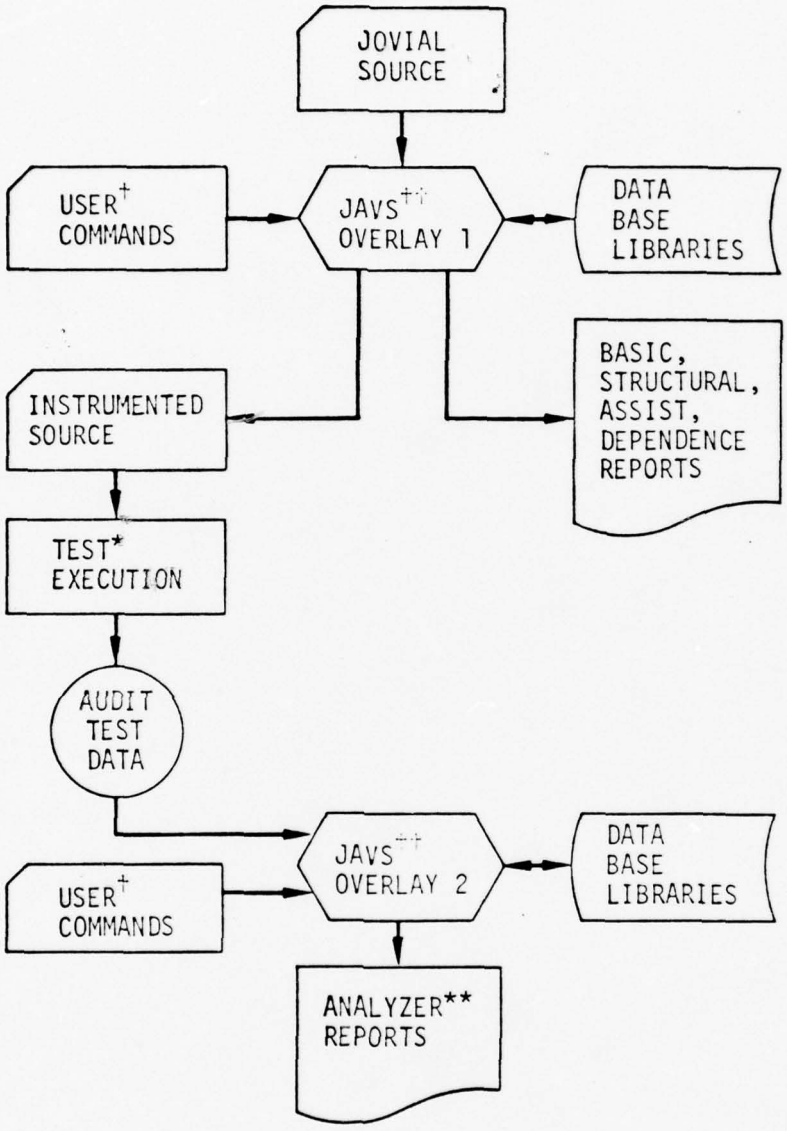


Figure 2.12. ANALYZER Processing

2.8 PROCESSING USING JAVS OVERLAY



† JAVS STANDARD AND/OR MACRO COMMANDS
* SEE FIG. 2.8
** REPORTS FROM ANY PROCESSING STEP EXCEPT BASIC
†† FOR CORE SIZE EFFICIENCY TWO JAVS OVERLAY PROGRAMS ARE USED: OVERLAY 1 IS THE COMPLETE SYSTEM; OVERLAY 2 DOES NOT CONTAIN SYNTAX ANALYSIS (BASIC)

3 JAVS CONSTRAINTS

JAVS imposes certain restrictions on the size of the database library, the command language, and the source text to be analyzed. Most of the limitations based on size are generous (e.g., the maximum number of nested IF statements is 100). Some of the limitations, such as those in the Test Execution process, can be raised by recompiling parts of JAVS. Some of the restrictions are based upon computer page dimensions and cannot be changed.

A number of the constraints which affect the incoming JOVIAL source code are founded in the principle of analyzing invokable modules separately. The dialects of JOVIAL recognized by JAVS allow certain constructs involving jumps to global labels, invocations of externally declared switches, and TEST statements where the FOR variable is external to the module. Some of these constructs require modification of the JOVIAL source code, and some cause a warning message to be issued to make the user aware of limitations in the DD-path test measurement report and execution tracing.

The constraints are listed in sections (Sec. 3.2 through 3.8) appropriate to a particular JAVS processing step. Universal constraints (Sec. 3.1) affect all of JAVS processing. The terminology used in describing the constraints requires knowledge of JOVIAL and JAVS. The user should refer to the JOCIT Compiler Users Manual[3] for JOVIAL terms and to the index in this manual for direction to the descriptions and references to JAVS terms.

3.1 UNIVERSAL CONSTRAINTS

1. Maximum 3 continuation cards for any given JAVS command.
2. Maximum 24 commas in any given JAVS command.
3. Maximum 150 modules selected for any given JAVS command iteration loop.
4. Maximum 250 word-pairs in a statement block.*
5. Maximum 100 statements on any DD-path.
6. Maximum 25 internal data base tables during any JAVS execution.
7. 100 JAVS errors produce a fatal error.
8. Maximum 250 known modules during any one JAVS execution (i.e., total modules in one or both libraries).
9. In order to change a previously made library, the name specified for an ALTER LIBRARY must be the same as when it was a CREATE LIBRARY.
10. Maximum 150 modules per JAVSTEXT.
11. Maximum 80 characters per card image read.
12. Maximum 128 characters per line of output.
13. No analysis is performed on a DIRECT code sequence.
14. The recognized dialects of JOVIAL include JOVIAL/J3, CDC JOVIAL, and Honeywell JOVIAL. The combination of these is processed by JAVS.

*BASIC automatically separates long source statements into a sequence of statements. The first statement is given the statement type according to the original JOVIAL source statement. Subsequent statements are of type continuation (CONT).

3.2 JAVS BASIC CONSTRAINTS

BASIC processing is capable of handling quite large source text files. Unusually large programs may have to be processed by several successive executions of BASIC, each operating on a separate file of START-TERM texts. In this event it is essential that JAVSTEXT directives are inserted in the source code prior to BASIC processing.

The following implementation constraints are the current ones which must be observed during BASIC processing:

1. Each module placed on the same library must have a unique module name for a given JAVSTEXT name. For this purpose only the first eight characters of any name are used. The first six characters should be unique (a compiler restriction).

2. COMPOOL texts should only be processed through BASIC.

3. If any errors are detected in JOVIAL source by BASIC, one or more statements on the library may be flagged as not parsed. The errors found in BASIC are summarized in Appendix B.

4. Hollerith and transmission type items must not exceed 120 characters.

5. Only one COMPOOL for each execution of BASIC can be placed on the library, when processing both COMPOOL and executable text. If one or more COMPOOLS are referenced by an executable text, the COMPOOL(S) need not be processed by BASIC, but the program text must contain a JAVSTEXT directive which indicates the presence of COMPOOL references.

3.3 JAVS STRUCTURAL CONSTRAINTS

1. A module should not GOTO a statement, directly or indirectly through a SWITCH, which is in another module.

2. A TEST statement must not appear in the range of a single factor FOR statement sequence.

3. Function calls with side effects (i.e., two successive calls to the function which produce different results) are not permitted in FOR, IF, IFEITH, and ORIF statements.

4. A declaration statement containing an END cannot appear as the last statement preceding the TERM statement of a program, procedure, or close.

5. The maximum depth of nesting of control or compound statements (IFEITH, ORIF, IF, FOR, BEGIN) is 100.

6. The maximum number of DD-paths which can begin at a statement is 100.

7. The maximum number of statements on a single DD-path is 100.

8. CLOSE invocations which appear as switch points in SWITCH declarations are treated as null switch points.

9. SWITCH invocations which appear as switch points (nested switches) in SWITCH declarations are treated as null switch points.

10. Invocations of externally declared switches are treated as RETURNs.

11. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR. If there are no three factor FOR statements in the parallel FOR, the first FOR statement is chosen as the controlling FOR for the purpose of constructing interstatement pointers in the JAVS statement descriptor blocks.

12. All programs, close and procedures must contain a BEGIN-END set of statements surrounding the executable code.

13. CLOSE declarations within a FOR statement may not use a TEST statement to reference the active FOR variable.

3.4 JAVS INSTRUMENT CONSTRAINTS

1. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR and is instrumented.

2. Invocations of externally declared switches are not instrumented.

3. The maximum number of nested IF statements is 100.

4. Subscript expressions in SWITCH invocations are limited to 72 characters.

5. Item names and switch names are limited to 30 characters in length.

6. The maximum number of variables allowed in a single TRACE directive is 18.

7. The maximum number of variables in TRACE directives is limited to 999 per module.

8. The maximum number of nested IFEITH and/or IFEITH/ORIF statements is 100.

9. The maximum DD-path number is 9999.

10. The maximum length of the TEXT in an ASSERT directive is 72 characters.

11. The functional modifiers BIT, CHAR, MANT, NENT, POS and BYTE may not appear in the numeric formulas for FOR statements; i.e., no side-effects are allowed in the initial value formula.

12. FOR variable may not appear in TRACE or EXPECT directives.

13. A declaration statement containing an END cannot appear as the last statement preceding the TERM statement of a program, procedure or close.

3.5 JAVS ASSIST CONSTRAINTS

1. Maximum of 100 DD-paths per reaching set path.
2. Maximum of 100 outways per decision.
3. Maximum of 1200 DD-paths per analyzed module for reaching set.
4. Maximum of 2400 statements per analyzed module for reaching set.
5. Maximum of 200 statements in reaching set.
6. Maximum 100 JAVSTEXTs specified for cross-reference mapping.
7. Maximum of 125 modules specified for cross-reference mapping.

3.6 TEST EXECUTION CONSTRAINTS

There is no probe test data recorded on AUDIT until after the first invocation of PROBI has been made. (PROBI invocations must be inserted manually or by using the INSTRUMENT, STARTTEST and INSTRUMENT, STOPTEST commands.) The PROBI invocation which indicates end-of-test must be executed to properly terminate data recording on AUDIT. When this end-of-test invocation is made, a message is issued.

There is a limitation of 100 instrumented modules in the workspace for the JAVS data collection routines. This limitation may be changed by recompiling the routines.

There is a limitation of 2000 instrumented DD-paths in the workspace for the JAVS data collection routines. This limitation may be changed by recompiling the routines.

3.7 JAVS DEPENDENCE CONSTRAINTS

1. Maximum 100 modules in each of the groups in group dependency analysis.

2. Maximum of 115 modules in sum of group A and group C in group dependency analysis.

3. Maximum bandwidth of 5 specified in BAND analysis.

3.8 JAVS ANALYZER CONSTRAINTS

In a single execution of ANALYZER, either MODTRACE or DDPTRACE option may be used, but not both.

Since ANALYZER does not reposition the file AUDIT at the conclusion of processing, two consecutive executions of ANALYZER in the same command set process different test cases on AUDIT.

Maximum of 100 test cases is analyzed.

If the test termination PROBI module is not invoked, ANALYZER will process the test case data on the AUDIT file up to the start of the last test case. When ANALYZER attempts to process the last test case on AUDIT and hits an end-of-file before the required 4H(TERM), written on AUDIT by the test termination call to PROBI, the error message "Illegal record on AUDIT file" will be issued. Thus, if there is only one test case on the AUDIT file and no occurrence of 4H(TERM), ANALYZER will have no summary data to process.

4 JAVS COMMANDS

JAVS processing is implemented by a command language (see Sec. 1.2). This section describes the eight basic types of commands. Each JAVS command starts with a keyword. The keywords are intended to be descriptive, concise, while maintaining flexibility through their processing options. Table 4.1 shows the relationship between the keywords and basic types of commands. A partial list is given below for process option, execution, and print/punch commands. Complete lists are given for the remaining types of commands.

Section 5 contains a complete description for each JAVS command. The commands in Sec. 5 are presented alphabetically.

- Library commands - (Sec. 4.1)

  OLD LIBRARY = <name>.

  CREATE LIBRARY = <name>.

  ALTER LIBRARY = <name>.

  DESCRIBE = <ON/OFF>.

  MERGE.

- Startup command - (Sec. 4.2)

  START.

- Module Selection commands - (Sec. 4.3)

  JAVSTEXT = <name>.

  MODULE = <name>.

  FOR LIBRARY.

  FOR JAVSTEXT.

  FOR MODULE = <name-1>,...,<name-n>.

  END FOR.

- Process Option commands (an incomplete list) - (Sec. 4.4)

  BASIC,COMMENTS = OFF.

  INSTRUMENT,MODE = FULL.

TABLE 4.1

KEYWORDS FOR COMMANDS

COMMAND TYPE	KEYWORD
Library	OLD LIBRARY CREATE LIBRARY ALTER LIBRARY DESCRIBE MERGE
Startup	START
Module Selection	JAVSTEXT MODULE FOR LIBRARY FOR JAVSTEXT FOR MODULE END FOR
Process Option and Process Execution	BASIC STRUCTURAL INSTRUMENT ASSIST DEPENDENCE ANALYZER
Print/Punch	PRINT PUNCH
Termination	END
Macro*	BUILD LIBRARY PROBE DOCUMENT TEST

*Available only with JAVS overlay version.

- Process execution commands (an incomplete list) - Sec. 4.5

  BASIC.

  STRUCTURAL.

  INSTRUMENT.

  ASSIST,CROSSREF,LIBRARY.

  DEPENDENCE,GROUP,LIBRARY.

  ANALYZER,SUMMARY.

- Standard print and punch commands (incomplete list)-Sec. 4.6

  PRINT,MODULE .

  PUNCH,JAVSTEXT=<name>.

  PRINT,DDPATHS.

- Run termination command - Sec. 4.7

  END.

- Macro command - Sec. 4.8

  {BUILD LIBRARY {=name}.

  PROBE {,options}.

  DOCUMENT {,options}.

  TEST {,options}.

4.1 LIBRARY COMMANDS

Three of the library commands (OLD LIBRARY, CREATE LIBRARY, and ALTER LIBRARY) define the name and status of the library (or libraries) during the current run. DESCRIBE command sets a flag to print the library manager statistics at the end of the run. The MERGE command causes the contents of the library specified in the OLD LIBRARY command to be written onto the library specified by CREATE or ALTER LIBRARY command.

4.1.1 Library Description Commands

- OLD LIBRARY = <name>. This option specifies whether LIBOLD is to be used in the current run, as is normally the case in all except BASIC and STRUCTURAL. The absence of this command indicates that no old library will be used. LIBOLD is a read-only library.

- CREATE LIBRARY = <name>. This option specifies that LIBNEW (a "new library") will be created during the current run, as is normally the case in BASIC. When this option is specified, LIBNEW is assumed to be empty at the beginning of a run. LIBNEW is used as a read/write library.

- ALTER LIBRARY = <name>. This option specifies that LIBNEW is a previously generated library to be modified during the current run, as is normally the case in STRUCTURAL. The name of LIBNEW should be the same one used when it was first created. LIBNEW is used as a read/write library.

These three library description commands specify two libraries during JAVS processing. There are five combinations of library commands for specifying JAVS libraries.

1. CREATE LIBRARY = <name>.

The only processing step for which LIBNEW alone is specified is BASIC. The overlay version of JAVS allows this library command to be used in multiple step processing without additional library definition. The overlay particulars are described along with other machine dependent information in the JAVS User's Guide.

2. OLD LIBRARY = <name>.

LIBOLD by itself is used whenever no new information is to be added to the library. It can be used in any processing step except BASIC. Processing steps ASSIST, DEPENDENCE and ANALYZER use this library; INSTRUMENT can use LIBOLD and save the instrumented file (LPUNCH) on disk, tape, or punched cards. This method of not storing the instrumented source in the library (by

not using ALTER or CREATE LIBRARY) is useful if the library is already large before instrumentation.

Although LIBNEW must be used for processing subsequent to STRUCTURAL, there may be instances where a preliminary STRUCTURAL run, using only LIBOLD, is desirable. JAVS printout reflects any database modifications made during a run; however, the modifications are stored only on LIBNEW (the read/write library). Thus LIBOLD remains unchanged.

3. ALTER LIBRARY = <name>.

This command modifies LIBNEW and saves the altered library. STRUCTURAL and INSTRUMENT data are often saved on the database library in this manner.

4. OLD LIBRARY = <name-1>.
   CREATE LIBRARY = <name-1>.

This set of library commands allows the creation of a new library by copying the contents of LIBOLD for the specified modules onto LIBNEW and then updating LIBNEW during the current run. LIBOLD remains unchanged.

5. OLD LIBRARY = <name-1>.
   ALTER LIBRARY = <name-2>.

This set of commands differs from the previous combination in several respects: (1) LIBNEW must be a previously created library, (2) for the specified modules already on LIBNEW, data from the current run is added to LIBNEW, and (3) modules previously only on LIBOLD are copied onto LIBNEW and updated by the current run. LIBOLD remains unchanged.

Either of the last two sets of library description commands can be used in any processing step when a library merge (see Sec. 4.1.3) is desired. They can also be in STRUCTURAL and INSTRUMENT when an unchanged library, LIBOLD, is desired as well as an updated LIBNEW.

All library description commands must precede the START command.

4.1.2 Library Access Command

- DESCRIBE = ON. or DESCRIBE = OFF. This option turns ON or OFF a sequence of printouts which describe in detail the access interface to the libraries. The default is OFF. This option is intended for JAVS maintenance. The DESCRIBE command must precede the START command.

4.1.3 Library Process Command

The MERGE command can be used (1) to add new modules to an existing library, and (2) to replace old modules with new modules having the same module name and text name. The command is

MERGE.

The MERGE command merges a LIBOLD with a LIBNEW. After merging LIBNEW contains all modules on the original LIBNEW, plus all modules on the LIBOLD that do not have the same module name and text name as a module on the LIBNEW. Both LIBOLD and LIBNEW must be specified in one of the two sets of library description commands (see Sec. 4.1.2).

The MERGE command must be preceded by the START command.

4.2 STARTUP COMMAND

The startup command terminates the library description and indicates the start of processing. The command is

START.

and causes the report in Fig. 4.1 to be printed. See command END in Sec. 5 for a description of the items in this report.

STARTER INITIALIZATION...

KNOWN MODULE DESCRIPTOR BLOCKS...

NO.	MODULE NAME	TEXT NAME	PARENT NAME	TYPE	PROC SCOPE	PROBE LINS	PROBE STMTS	EXEC STMTS	FIRST EXEC	WORD PAIRS	TOKS	SYMS	SLTS	DMTS	DDPS	DS BLKS	PARMS IN	PARMS OUT	DIRECT CODE	COMP TOT	COMP DDP
1	EXCOMPL	EXCOMPL	EXCOMPL	CMPL		0	13	0	0	102	53	0	6	0	0	0	0	0	NO	0	0
2	EXPROGM	EXPROGM	EXPROGM	PROG	EXT	0	33	14	16	270	192	2	26	2	5	23	0	0	NO	0	0
3	EXMPL2	EXPROGM	EXPROGM	CLSM	GLBL	0	6	5	4	24	16	1	4	0	1	5	0	0	NO	0	0
4	EXMPL1	EXPROGM	EXPROGM	PROC	INT	0	44	29	9	279	202	3	21	1	20	84	2	0	NO	0	0
5	EXMPL3	EXPROGM	EXMPL1	CLSP	LOCL	0	5	4	4	21	14	1	3	0	1	4	0	0	NO	0	0

LIBRARY INFORMATION--RUN OF 0731

NO.	LIBRARY NAME	TYPE ACCESS	DATE CREATED	TIMES ALTERED	LAST ALTERED	TOTAL WORDS	LIBRARY MODULES	LIBRARY FRAGMENTS
1	TEST	READ	0731	2	0731	17520	5	35
2	SCRATCH	NEW	0731	1	0731	20	0	3

Figure 4.1.

4.3 MODULE SELECTION COMMANDS

Many JAVS commands require specification of the particular module upon which computations are to be performed. [Some of the standard print commands require this, for example.] If both LIBNEW and LIBOLD have been declared, the module selector will first look in LIBNEW for the module and, if not found, will look in LIBOLD. Only the version on LIBNEW will be selected if a module with the same name appears on both libraries. If two or more versions of a module appear on a library, the last one entered on the library will be selected.

A module is identified by its START-TERM text name and its module name. The text name is taken from the JAVSTEXT directive which appears at the beginning of the START-TERM text sequence. If there is no JAVSTEXT directive, the name *NOJAVS* is automatically supplied by BASIC. The module name is taken from the JOVIAL statement which identifies the module (e.g., program,* PROC, CLOSE, COMPOOL). Only the first eight characters of the name on the JOVIAL statement are used for the module name. The first eight characters of the name on the JAVSTEXT are used as the text name (Sec. 1.4). The first six characters of each JAVSTEXT on the library must be unique.

4.3.1 START-TERM Text Selection

Some JAVS commands require specification of the START-TERM text sequence upon which computations are to be performed. START-TERM texts are known to JAVS by their names. The following command makes the "current text" be the named one:

JAVSTEXT = <text-name>.

All subsequent commands (if they refer to a specific text) are applied to the specified text. There can be any number of JAVSTEXT= commands. A new JAVSTEXT= command replaces and supersedes a previous JAVSTEXT= command.

4.3.2 Single Module Selection

Modules are known to JAVS by their names and the text to which they belong. The following makes the "current module" within the "current text" be the named one:

MODULE = <name>.

All subsequent commands (if they refer to a specific module) are applied to the specified module within the "current text." If no "current text" has been specified, the module selector will locate the module using only its name. There can be any number of MODULE= commands.

---

*Some JOVIAL compilers require that the main program be specified with a PROGRAM statement.

4.3.3 Multiple Module Iteration

In many applications, the user will want to repeat a command for a number of different modules. The three forms of command iteration structure are described below.

4.3.3.1 Selected Module Iteration.

The following sequence selects a number of modules, by name, and iterates a block of commands (which cannot contain another iteration) once for each specified module:

```
FOR MODULE = <name-1>,<name-2>,...,<name-n>.

(any set of commands)

END FOR.
```

4.3.3.2 ALL-Modules Iteration for Specified JAVSTEXT (START-TERM Text).

The following sequence selects each known module within the "current text" and iterates a block of commands (which cannot contain another iteration) once for each known module of that text:

```
FOR JAVSTEXT.

(any set of commands)

END FOR.
```

4.3.3.3 ALL-Modules Iteration for Library.

The following sequence selects each known module on the library and iterates a block of commands (which cannot contain another iteration) once for each known module:

```
FOR LIBRARY.

(any set of commands)

END FOR.
```

4.3.4 Module Selection Constraint

The maximum number of modules which may be specified by a single iteration is 150 (see Sec. 3).

4.4 PROCESS OPTION COMMANDS

Processing steps BASIC, STRUCTURAL, INSTRUMENT and ANALYZER have option commands which define the action taken when the process execution command is given. The option commands follow the START command and precede the appropriate execution command (see Sec. 4.5).

4.4.1 BASIC Option Commands

The BASIC options are specified by the following commands (with the default value assumed in case the option command is not given prior to the appearance of the BASIC execution command):

BASIC,CARD IMAGES = ON/OFF.	DEFAULT = ON.
BASIC,COMMENTS = ON/OFF.	DEFAULT = ON.
BASIC,DEFINES = ON/OFF.	DEFAULT = ON.
BASIC,ERRORS = ON/OFF/LIMIT/TRACE.	DEFAULT = ON.
BASIC,SYMBOLS = ON/OFF/PARTIAL.	DEFAULT = PARTIAL.
BASIC,TEXT = PRESET/COMPUTE/BOTH/JAVSTEXT.	DEFAULT = COMPUTE.

BASIC options which have been selected in the command sequence remain in effect until they are reset by a later command. See Sec. 5 for a complete definition and example of each BASIC option command.

The user will note that module selection commands do not apply until after a module has been added to a library during BASIC processing by the BASIC verb. The name assigned to a module when it is added to a library is the first eight characters of the program name, procedure name, or close name of the module. The first six characters of the names should be unique.

It is important that the command sequence involving one or more occurrences of the BASIC execution command be terminated with the END command so that LIBNEW is closed properly. The user may employ the standard print and punch commands (Sec. 5) after the BASIC command and after module or JAVSTEXT specification (Sec. 4.3).

4.4.2 STRUCTURAL Option Command

The STRUCTURAL print option is specified by the following command (with the specified default value assumed in case the command is not given prior to the appearance of the STRUCTURAL verb):

STRUCTURAL,PRINT = SUMMARY/DEBUG. (DEFAULT = SUMMARY)

The STRUCTURAL option which has been selected in the command sequence remains in effect until it is reset by a later option command. Each instance of the STRUCTURAL execution command in a command sequence assumes that a module has been selected, and that the STRUCTURAL execution command will be applied only once to this module.

See Sec. 5 for a definition and example of the STRUCTURAL print option.

4.4.3 INSTRUMENT Option Commands

The INSTRUMENT options are specified by the following commands (with the specified default value assumed in case the command is not given prior to the appearance of the INSTRUMENT execution command):

```
INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.
                                                DEFAULT = DDPATHS.
INSTRUMENT,PROBE,DDPATH = <probe-name>.     DEFAULT = PROBE*
INSTRUMENT,PROBE,MODULE = <invocation-name>.
                                                DEFAULT = PROBM*
INSTRUMENT,PROBE,TEST = <test-name>.        DEFAULT = PROBI*
```

INSTRUMENT options which have been selected in the command sequence remain in effect only during INSTRUMENT processing of the current module. Thus if the same option is requested for a group of modules, the INSTRUMENT option and INSTRUMENT execution command should be placed within the module selection iteration (see Sec. 4.3.3).

4.4.4 ANALYZER Option Commands

The ANALYZER options are specified by the following commands. There are few default options for ANALYZER, since most option commands produce one or more reports.

Unlike the other processing steps, ANALYZER has its own module selection which does not use a JAVSTEXT specification. If there are several modules with the same name, but with different JAVSTEXT names, coverage reports should be specified by the option:

---

*PROBE is an arbitrary name used for the DD-path data collection routine.
PROBM is an arbitrary name used for the module-invocation-and-return data collection routine.
PROBI is an arbitrary name used for the test-beginning-and-test-end data collection routine.
PROBD is an arbitrary name for the description writing routine.

ANALYZER, ALL MODULES.

If not all of the modules were instrumented, they will be noted as "not traced."

The ANALYZER options are of the following format:

```
ANALYZER,MODULE = <name-1>,...,<name-n>.
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER,HIT.
ANALYZER,NOTHIT.
ANALYZER,MODLST.
ANALYZER,MODTRACE.
ANALYZER,DDPTRACE.
ANALYZER,TIME.
ANALYZER,DDPATHS.
ANALYZER,CASES = <number>.                    DEFAULT = 10.
ANALYZER,FACTOR = <percent increase>.         DEFAULT = 0.
ANALYZER,ALL.
```

A description and example for each option is given in Sec. 5.

4.5 PROCESS EXECUTION COMMANDS

For the four processing steps described in the option commands, the process execution commands are:

```
BASIC.

STRUCTURAL.

INSTRUMENT.

ANALYZER.
```

Each of these commands causes execution of the processing step on the specified modules (except for BASIC, where modules are not specified) according to the previously specified options or their default values.

The ASSIST command set contains only execution commands. Module selection must be made for these commands.

```
ASSIST,REACHING SET,<specifications>.

ASSIST,CROSSREF,<specifications>.

ASSIST,PICTURE[,<options>].

ASSIST,STATEMENTS.
```

The DEPENDENCE command set contains only execution commands; some commands require module specification (see Sec. 4.3.2) and some commands operate on the entire library. (See Sec. 5 for a detailed description of each command.) All DEPENDENCE commands produce a report based on intermodule dependencies.

The DEPENDENCE module commands are:

```
DEPENDENCE,PRINT,INVOKES.

DEPENDENCE,BANDS = <number>. (DEFAULT = 2)

DEPENDENCE,TREE.
```

The DEPENDENCE library commands are:

```
DEPENDENCE,GROUP MODULES = <name-1>,...,<name-n>.

DEPENDENCE,GROUP,LIBRARY

DEPENDENCE,GROUP,AUXLIB.

DEPENDENCE,SUMMARY.
```

4.6 STANDARD PRINT AND PUNCH COMMANDS

4.6.1 PRINT

The standard print commands provide the means to generate formatted renditions of JAVS internal tables. These print commands are universal; i.e., they can be used in any processing step. Ordinarily, these commands are used after STRUCTURAL to print the key tables generated during BASIC and STRUCTURAL. The standard print commands are of the form:

PRINT, <table-name>.

PRINT,JAVSTEXT=<name>[, options].

where **<table-name>** is DDP, DDPATHS, DMT, MODULE, SB, SDB, SLT, STB or SYMTAB. For a PRINT table to be accepted, a module must have been selected with a module selection command (see Sec. 4.3.2). The PRINT JAVSTEXT commands are used to display the START-TERM text as it is presented to the JOVIAL compiler. The options specify instrumented or uninstrumented source text.

4.6.2 PUNCH

The standard punch commands provide the means to generate reformatted versions of the source text in preparation for the JOVIAL compiler. The source text is written onto file LPUNCH (see Table 1.1 in Sec. 1.1.3). The standard punch commands are of the form:

PUNCH,MODULE.

PUNCH,JAVSTEXT=<name>[, options].

The PUNCH MODULE command requires module selection to be made prior to the command. The JAVSTEXT punch command is used to prepare the complete START-TERM text for compilation. The options specify whether the text is to be output to LPUNCH in its instrumented or uninstrumented form.

A complete description of each standard print and punch command is given in Sec. 5.

4.7 RUN TERMINATION COMMAND

A JAVS run terminates on the END command, which provides for correctly closing any files and writing a wrapup summary. If the wrapup report, shown in Fig. 4.2, is not printed, then LIBNEW was not closed properly and cannot be used in subsequent runs.

The run termination command is

```
END.
```

```
JAVS WRAPUP...

KNOWN MODULE DESCRIPTOR BLOCKS...
```

NO.	MODULE NAME	TEXT NAME	PARENT NAME	TYPE	PROC SCOPE	PROBE LINS	PROBE STMTS	EXEC FIRST STMTS	WORD EXEC PAIRS	WORD PAIRS	TOKS	SYMS	SLTS	DMTS	DOPS	DS BLKS	PARMS IN	PARMS OUT	DIRECT CODE	COMP TOT	COMP DOP
1	EXCOMPL	EXCOMPL	EXCOMPL	CMPL		0	13	0	0	102	53	0	6	0	0	0	0	0	40	0	0
2	EXPROGM	EXPROGM	EXPROGM	PROG	EXT	0	33	14	16	270	192	2	26	2	5	23	0	0	40	0	0
3	EXMPL2	EXPROGM	EXPROGM	CLSM	GLBL	0	6	5	4	24	16	1	4	0	1	5	0	0	40	0	0
4	EXMPL1	EXPROGM	EXPROGM	PROC	INT	0	44	29	9	279	202	3	21	1	20	84	2	0	40	0	0
5	EXMPL3	EXPROGM	EXMPL1	CLSP	LOCL	0	5	4	4	21	14	1	3	0	1	4	0	0	40	0	0

```
LIBRARY INFORMATION--RUN OF 0731
```

NO.	LIBRARY NAME	TYPE ACCESS	DATE CREATED	TIMES ALTERED	LAST ALTERED	TOTAL WORDS	LIBRARY MODULES	LIBRARY FRAGMENTS
1	**NOT OPENED**							
2	TEST	R/W	0731	2	0731	17520	5	35

Figure 4.2.

5 JAVS COMMANDS

This section contains a complete list of JAVS commands, in alphabetical order, along with the JAVS processing step in which each command is used. The term "universal" indicates that the command can be used in any processing step.*

Following the list of commands is a description, accompanied with sample output and command sets, of each JAVS command.

*The overlay version of JAVS allows all commands to be "universal."

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
ALTER LIBRARY = <libname>.	(Universal)
ANALYZER.	ANALYZER
ANALYZER,ALL.	ANALYZER
ANALYZER,ALL MODULES.	ANALYZER
ANALYZER,CASES = <number>.	ANALYZER
ANALYZER,DDPATHS.	ANALYZER
ANALYZER,DDPTRACE.	ANALYZER
ANALYZER,FACTOR = <percent-increase>.	ANALYZER
ANALYZER,HIT.	ANALYZER
ANALYZER,MODLST.	ANALYZER
ANALYZER,MODTRACE.	ANALYZER
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.	ANALYZER
ANALYZER,NOTHIT.	ANALYZER
ANALYZER,SUMMARY.	ANALYZER
ANALYZER,TIME.	ANALYZER
ASSIST,CROSSREF,JAVSTEXT = <text-name-1>,<text-name-2>,..., <text-name-n>.	ASSIST
ASSIST,CROSSREF,LIBRARY.	ASSIST
ASSIST,PICTURE.	ASSIST
ASSIST,PICTURE{,CONTROL}{,NOSWITCH}.	ASSIST
ASSIST,REACHING SET,<number-to>{,<number-from>} {,PICTURE{,ITERATIVE}}.	ASSIST
ASSIST,STATEMENTS.	ASSIST
BASIC.	BASIC
BASIC,CARD IMAGES = ON/OFF.	BASIC
BASIC,COMMENTS = ON/OFF.	BASIC
BASIC,DEFINES = ON/OFF.	BASIC
BASIC,ERRORS = ON/OFF/LIMIT/TRACE.	BASIC
BASIC,SYMBOLS = ON/OFF/PARTIAL.	BASIC
BASIC,TEXT = COMPUTE/BOTH/PRESET/JAVSTEXT.	BASIC
*BUILD LIBRARY {= <library name>},	BASIC, STRUCTURAL
CREATE LIBRARY = <libname>.	(Universal)

*Can be used only with the overlay version.

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
DEPENDENCE,BANDS.	DEPENDENCE
DEPENDENCE,BANDS = <number>.	DEPENDENCE
DEPENDENCE,GROUP,AUXLIB.	DEPENDENCE
DEPENDENCE,GROUP,LIBRARY.	DEPENDENCE
DEPENDENCE,GROUP,MODULES = <name-1>,<name-2>,...,<name-n>.	DEPENDENCE
DEPENDENCE,PRINT,INVOKES.	DEPENDENCE
DEPENDENCE,SUMMARY.	DEPENDENCE
DEPENDENCE,TREE.	DEPENDENCE
DESCRIBE = ON/<u>OFF</u>.	(Universal)
*DOCUMENT{,JAVSTEXT=<text-name>{,MODULE=<name-1>,...}}.	ASSIST, DEPENDENCE, (Universal)
END.	(Universal)
END FOR.	(Universal)
FOR JAVSTEXT.	(Universal)
FOR LIBRARY.	(Universal)
FOR MODULE = <name-1>,<name-2>,...,<name-n>.	(Universal)
INSTRUMENT.	INSTRUMENT
INSTRUMENT,MODE = INVOCATION/<u>DDPATHS</u>/DIRECTIVES/FULL.	INSTRUMENT
INSTRUMENT,PROBE,DDPATH = <probe-name>.	INSTRUMENT
INSTRUMENT,PROBE,MODULE = <invocation-name>.	INSTRUMENT
INSTRUMENT,PROBE,TEST = <test-name>.	INSTRUMENT
INSTRUMENT,STARTTEST = <modname>,<textname>,<stmt. no.> {,<TESNAM>}{,<TFLAG>}.	INSTRUMENT
INSTRUMENT,STOPTEST = <modname>,<textname>, <stmt. no.>.	INSTRUMENT
JAVSTEXT = <text-name>.	(Universal)
MERGE.	(Universal)
MODULE = <name>.	(Universal)
OLD LIBRARY = <libname>.	(Universal)

*Can be used only with the overlay version.

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
PRINT,DDP.	(Universal)
PRINT,DDPATHS.	(Universal)
PRINT,DMT.	(Universal)
PRINT,JAVSTEXT = <text-name-1>, INSTRUMENTED = ALL.	(Universal)
PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PRINT,JAVSTEXT = <text-name>.	(Universal)
PRINT,MODULE.	(Universal)
PRINT,SB.	(Universal)
PRINT,SDB.	(Universal)
PRINT,SLT.	(Universal)
PRINT,STB.	(Universal)
*PROBE{,JAVSTEXT = <text-name>{,MODULE = <name-1>,...}}.	INSTRUMENT, (Universal)
*PROBI,STARTTEST = <modname>,<textname>,<stmt. no.> {,TESNAM}{,TFLAG}.	INSTRUMENT, (Universal)
*PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.	INSTRUMENT, (Universal)
PUNCH,JAVSTEXT = <text-name>.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = ALL.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PUNCH,MODULE.	(Universal)
START.	(Startup)
STRUCTURAL.	STRUCTURAL
STRUCTURAL,PRINT = SUMMARY/DEBUG.	STRUCTURAL
*TEST{,MODULE = <name-1>,<name-2>,...<name-n>}.	ANALYZER, (Universal)

*Can be used only with the overlay version.

ALTER LIBRARY = <name>.

ALTER LIBRARY = <name>.

Description

This option specifies that LIBNEW is a previously generated library to be modified during the current run. LIBNEW is used as a read/write library.

Rule

The name of LIBNEW must be the same one used when it was first created.

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

This command set will modify the library created by the BASIC processing step and add the STRUCTURAL data to the library for each module on the library.

(2)
```
ALTER LIBRARY = REFONE.
START.
BASIC.
END.
```

This command set augments a previously built library with the BASIC processing of additional source text.

ANALYZER.

Description

The ANALYZER execution command causes processing of the post-test analysis. If no ANALYZER report options are present, no report is produced.

Rules

For a single ANALYZER command,

(1) Either option MODTRACE or DDPTRACE can be used, but not both.

(2) A maximum of 100 test cases is analyzed.

(3) The ANALYZER command does not reposition the AUDIT file; thus consecutive ANALYZER commands process different numbers of test cases, dependent upon the ANALYZER,CASES command.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,ALL.
ANALYZER.
END.
```

This set of commands will produce the SUMMARY, HIT, NOTHIT, TIME, DDPATHS, MODLST, DDPTRACE post-test analysis reports for all modules on the library.

(2)
```
OLD LIBRARY = TRFMAN.
START.
ANALYZER,CASES = 50.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,DDPATHS.
ANALYZER,DDPTRACE.
ANALYZER.
END.
```

ANALYZER,ALL.

Description

This command is equivalent to the following set of commands:

```
ANALYZER,SUMMARY.
ANALYZER,HIT.
ANALYZER,NOTHIT.
ANALYZER,TIME.
ANALYZER,DDPATHS.
ANALYZER,MODLST.
ANALYZER,DDPTRACE.
```

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN1.
START.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,ALL.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN1.
START.
ANALYZER,ALL MODULES.
ANALYZER,CASES = 100.
ANALYZER,ALL.
ANALYZER.
END.
```

ANALYZER,ALL MODULES.

Description

The module specification commands are used to identify the modules on LIBOLD for which ANALYZER processes the summary data on the Test Execution trace file. The default is no modules specified. To specify all known modules, use the command:

```
ANALYZER,ALL MODULES.
```

To specify a subset of modules, use the command:

```
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.
```

The following ANALYZER options require module specification: SUMMARY, HIT, NOTHIT, TIME, MODLST, DDPATHS, ALL.

Example Command Sets

(1)
```
OLD LIBRARY = REFTEST.
START.
ANALYZER,ALL MODULES.
ANALYZER,ALL.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,FACTOR = 20.
ANALYZER,TIME.
ANALYZER,ALL MODULES.
ANALYZER.
END.
```

ANALYZER,CASES = <number>. ANALYZER,CASES = <number>.

Description

This command is used to specify the maximum number of test cases to process from the test file. The absence of this command specifies the default of 10 test cases. A test case is determined by each invocation of the JAVS data collection routine PROBI with the second parameter, TFLAG, equal to 1, 2, or 3 (see Sec. 2.6.2.2). ANALYZER reads and analyzes the data for each test case encountered on the test file. At the conclusion of the specified number of test cases, or at the end of the last test case if less than the number specified, the ANALYZER reports are produced and all further analysis of the test file is inhabited. The maximum number of test cases which ANALYZER can process is 100.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,CASES = 2.
ANALYZER,SUMMARY.
ANALYZER,MODULE = EXMPL1.
ANALYZER.
END.
```

This command set will produce the post-test summary report for only the first two testcases for module EXMPL1, even if there are more than two test cases on the test file.

(2)
```
OLD LIBRARY = REF.
START.
ANALYZER,CASES = 100.
ANALYZER,ALL MODULES.
ANALYZER,ALL.
ANALYZER.
END.
```

This command set raises the number of allowable test cases from the default of 10 to the maximum limit of 100.

ANALYZER,DDPATHS.

Description

For each specified module, this command causes ANALYZER to produce a source listing for the statements which begin a DD-path, showing the DD-path description and an execution count reflecting DD-path coverage for all test-cases encountered on the trace file. The report is similar in format to the report from PRINT, DDPATHS, with execution counts appended. DD-path coverage is summarized at the end of the report. If a specified module is not invoked during the test, this report is suppressed.

Sample Output

```
MODULE DD-PATH COVERAGE LISTING

MODULE <EXAMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL          STATEMENT                                   DD-PATHS GENERATED                                  COVERAGE
-------------------------------------------------------------------------------------------------------------------------
   1 ( 0)          PROC EXAMPL1 ( LIMIT1 , LIMIT2 ) $
                                                               ** DD-PATH   1 IS PROCEDURE ENTRY                    2
                     .  .  .
   9 ( 1)            IF LIMIT1 GQ 100 $
                                                               ** DD-PATH   2 IS TRUE BRANCH                       0 *
                                                               ** DD-PATH   3 IS FALSE BRANCH                      2
                     .  .  .
  12 ( 1)            FOR I = 1 , 1 , LIMIT1 $
  13 ( 2)              BEGIN
                       .  .  .
  18 ( 2)              FOR J = 1 , 1 , LIMIT2 $
  19 ( 3)                BEGIN
                         .  .  .
  21 ( 3)                IFEITH J LQ 3 $
                                                               ** DD-PATH   4 IS TRUE BRANCH                       2
                                                               ** DD-PATH   5 IS FALSE BRANCH                      0 *
                         .  .  .
  23 ( 3)                ORIF 1 $
                                                               ** DD-PATH   6 IS TRUE BRANCH                       0 *
                         .  .  .
  25 ( 3)                END
                         .  .  .
  27 ( 3)                SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
  28 ( 3)                GOTO PICK (S INDXS - 1 S) $
                                                               ** DD-PATH   7 IS SWITCH OUTWAY   1    2
                                                               ** DD-PATH   8 IS SWITCH OUTWAY   2    0 *
                                                               ** DD-PATH   9 IS SWITCH OUTWAY   3    0 *
                                                               ** DD-PATH  10 IS SWITCH OUTWAY   4    0 *
                                                               ** DD-PATH  11 IS SWITCH OUTWAY   5    0 *
                         .  .  .  .
  30 ( 3)          LABEL1.
                         IFEITH RESULT LS 4 $
                                                               ** DD-PATH  12 IS TRUE BRANCH                       0 *
                                                               ** DD-PATH  13 IS FALSE BRANCH                      2
                         .  .  .
  32 ( 3)                ORIF RESULT EQ 4 $
                                                               ** DD-PATH  14 IS TRUE BRANCH                       0 *
                                                               ** DD-PATH  15 IS FALSE BRANCH                      2
                         .  .  .
  34 ( 3)                ORIF 1 $
                                                               ** DD-PATH  16 IS TRUE BRANCH                       2
                         .  .  .
  36 ( 3)                END
                         .  .  .
  39 ( 3)                END
                                                               ** DD-PATH  17 IS LOOP ON FOR AGAIN                 0 *
                                                               ** DD-PATH  18 IS ESCAPE FOR LOOP                   2
                       .  .  .
  41 ( 2)              END
                                                               ** DD-PATH  19 IS LOOP ON FOR AGAIN                 0 *
                                                               ** DD-PATH  20 IS ESCAPE FOR LOOP                   2
-------------------------------------------------------------------------------------------------------------------------

TOTAL DDPATHS              20
DDPATHS EXECUTED            9
PERCENT EXECUTED           45
```

ANALYZER,DDPATHS. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,DDPATHS.
ANALYZER,MODLST.
ANALYZER.
END.
```

This command set will produce only the two cumulative coverage reports, DD-path coverage and statement coverage, from the set of test cases on the trace file.

(2)
```
OLD  LIBRARY = REFMAN.
START.
ANALYZER,MODULE = EXAMPL1.
ANALYZER,DDPATHS.
ANALYZER,NOTHIT.
ANALYZER.
END.
```

This command set will produce the DD-path cumulative coverage report and the test-by-test report of DD-paths not executed for module EXMPL1.

ANALYZER,DDPTRACE.

Description

This command causes a printout of the Test Execution trace data at the DD-path level with the module invocation and return trace data interspersed. The trace data must have been recorded at the DD-path level; i.e., the TFLAG parameter in the PROBI call must be 3. Each line of output represents a module invocation or return, the start of a execution, or the beginning or end of a test case or test set. Each DD-path line shows the DD-path number, its description and outcome, and the relevant source text. (The FOR statement is considered to be the relevant statement in iteration.)

Rule

For a single ANALYZER command, either option DDPTRACE or MODTRACE can be used, but not both.

Sample Output Excerpt

```
                TEST-CASE                SAMPLE   2    2209  08/04/75
-------------------------------------------------------------------------------------------------
      STMT   19     IF READER NQ V(EOF) $                          *DD-PATH   2    TRUE BRANCH

** INVOKED MODULE IS EXMPL1  (EXPROGM )
      STMT    1     PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $              *DD-PATH   1    PROCEDURE ENTRY
      STMT    9     IF LIMIT1 GQ 100 $                             *DD-PATH   3    FALSE BRANCH
      STMT   21     IFEITH J LQ 3 $                                *DD-PATH   4    TRUE BRANCH
      STMT   29     GOTO PICK (S INDAS - 1 S) $                    *DD-PATH   7    SWITCH OUTWAY    1
      STMT   32     ORIF RESULT EQ 4 $                             *DD-PATH  15    FALSE BRANCH
      STMT   30 L466L1:                                            *DD-PATH  13    FALSE BRANCH
                    IFEITH RESULT LS 4 $
      STMT   34     ORIF 1 $                                       *DD-PATH  16    TRUE BRANCH
      STMT   18     FOR J = 1 , 1 , LIMIT2 $                       *DD-PATH  18    ESCAPE FOR LOOP
      STMT   12     FOR I = 1 , 1 , LIMIT1 $                       *DD-PATH  20    ESCAPE FOR LOOP
** RETURN FROM MODULE EXMPL1  (EXPROGM )

      STMT   26     IF ITER1 GQ 100 $                              *DD-PATH   5    FALSE BRANCH
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,DDPTRACE.
ANALYZER,DDPATHS.
ANALYZER.
END.
```

This command set will produce the DD-path trace report and the cumulative DD-path coverage report for all modules on the library.

ANALYZER,FACTOR = <percent>.

ANALYZER,FACTOR = <percent>.

Description

This command is used in conjunction with the ANALYZER,TIME. option. It allows the user to establish a value for the estimated CPU time overhead of executing the inserted probe statements in instrumented modules.

In the report produced by the command

```
ANALYZER,TIME.
```

both the actual time including the inserted probe statements and the estimated time using the <percent> specified is shown. Absence of the ANALYZER,FACTOR option command indicates no overhead in the time; i.e., the default is 0 percent.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,FACTOR = 20.
ANALYZER,TIME.
ANALYZER,ALL.
ANALYZER,ALL MODULES.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,FACTOR = 10.
ANALYZER,TIME.
ANALYZER.
END.
```

ANALYZER,HIT.

Description

This command causes ANALYZER to produce a list of DD-paths for each module and the number of times each path has been traversed. Two such lists are made: one for each test case and one for all test cases analyzed. Only the modules specified by an ANALYZER module selection option and encountered on the AUDIT file will be represented in this report.

Sample Output

```
DD-PATHS TRAVERSED IN SELECTED MODULES FOR TEST-CASE SAMPLE     2  2209  08/04/75

MODULE    JAVSTEXT  DD-PATH (TRAVERSALS)
-------------------------------------------------------------------------------
EXPROGM   EXPROGM     1(      1)   2(      1)   3(      0)   4(      0)   5(      1)
EXMPL1    EXPROGM     1(      1)   2(      0)   3(      1)   4(      1)   5(      0)   6(      0)   7(      1)
                      8(      0)   9(      0)  10(      0)  11(      0)  12(      0)  13(      1)  14(      0)
                     15(      1)  16(      1)  17(      0)  18(      1)  19(      0)  20(      1)
```

Single Test Case

```
DD-PATHS TRAVERSED IN SELECTED MODULES FOR ALL TEST-CASES

MODULE    JAVSTEXT  DD-PATH (TRAVERSALS)
-------------------------------------------------------------------------------
EXPROGM   EXPROGM     1(      1)   2(      2)   3(      1)   4(      0)   5(      2)
EXMPL1    EXPROGM     1(      2)   2(      0)   3(      2)   4(      2)   5(      0)   6(      0)   7(      2)
                      8(      0)   9(      0)  10(      0)  11(      0)  12(      0)  13(      2)  14(      0)
                     15(      2)  16(      2)  17(      0)  18(      2)  19(      0)  20(      2)
```

Multiple Test Cases

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,HIT.
ANALYZER.
END.
```

Description

This command causes ANALYZER to print a source listing of each specified module which shows the number of times each executable statement was exercised for all testcases. The DD-paths are listed at the right of the statement of origin. Next is a count of the times each executable statement was exercised; executable statements not exercised are flagged with an asterisk. To the far right are control indicators for statement types which affect normal processing sequence. Statement coverage is summarized at the end of the module listing. If a specified module is not invoked during the test, this report is suppressed.

Sample Output

```
MODULE STATEMENT LISTING

MODULE <EXPROGM >, JOVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL          STATEMENT                                                    DD-PATHS          CONTROL
-------------------------------------------------------------------------------------------------------
   1  ( 0)        **, JOVSTEXT EXPROGM COMPUTE ( EXCOMPL ) **
   2  ( 0)        START
   3  ( 0)        $
   4  ( 0)        ** JOVIAL SIMPLE TEST PROGRAM **
   5  ( 0)        ITEM ID H 4 $
   6  ( 0)        ITEM ITER1 I 24 S $
   7  ( 0)        ITEM ITER2 I 24 S $
   8  ( 0)        ITEM ITER1A H 4 $
   9  ( 0)        ITEM ITER2A H 4 $
  10  ( 0)        OVERLAY ITER1 = ITER1A $
  11  ( 0)        OVERLAY ITER2 = ITER2A $
  12  ( 0)        ITEM CARD H 80 $
  13  ( 0)        FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
  14  ( 0)        FILE PRINTR H 0 R 12A V(OK) V(EOF) TAPE6 $
                                                                                                  0 *
  15  ( 0)        MONITOR ID , ITER1A , ITER2A $
                                                                                                  1
  16  ( 0)        MESSAG = 'MSG1 $                                                 (  1)          1
  17  ( 0)        OUTPUT PRINTR MESSAG $                                                          1      I/O
                                                                                                  3      I/O <------
  18  ( 0)    BG.
              INPUT READER CARD $
  19  ( 0)        IF READER NO V(EOF) $                                            (  2=  3)      3      IF
  20  ( 1)          BEGIN                                                                         2
  21  ( 1)          BYTE (S 0 , 4 S) ( ID ) = BYTE (S 0 , 4 S) ( CARD ) $                         2
  22  ( 1)          BYTE (S 0 , 4 S) ( ITER1A ) = BYTE (S 9 , 4 S) ( CARD ) $                     2
  23  ( 1)          BYTE (S 0 , 4 S) ( ITER2A ) = BYTE (S 19 , 4 S) ( CARD ) $                    2
  24  ( 1)          EXMPL1 ( ITER1 , ITER2 ) $                                                    2      INV
  25  ( 1)        ** EXMPL2 **
  26  ( 1)          IF ITER1 GO 100 $                                              (  4=  5)      2      IF
  27  ( 2)            GOTO EXMPL2 $                                                               0 *    INV
  28  ( 1)          GOTO BG $                                                                     2      ------>
  29  ( 1)          END                                                                           0 *
  30  ( 0)        ** IF **
                                                                                                  1
  31  ( 0)        STOP $
  32  ( 0)        ** EXMPL1 **
  33  ( 0)        TERM $
-------------------------------------------------------------------------------------------------------

EXECUTABLE STATEMENTS      14
STATEMENTS EXECUTED        12
  PER CENT EXECUTED        85
```

ANALYZER,MODLST. (Cont.)

Example Command Set

```
OLD LIBRARY = REFMAN.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,MODLST.
ANALYZER.
END.
```

Description

This command causes a printout of the Test Execution trace data at the module invocation and return level. If the trace data was not recorded on the file during Test Execution (see Sec. 2.6.2.2), no report is produced. Each line of output represents either a module invocation or the beginning or end of a test case or test set. The level of invocation for the modules being traced is shown to the left; the module invocations are indented to show nesting level.

Rule

For a single execution of ANALYZER, either option MODTRACE or DDPTRACE can be used, but not both.

Sample Output

```
-------------------------------------------------------------------------------
                    TEST-CASE                SAMPLE   1          1049     08/05/75
-------------------------------------------------------------------------------
(   0)   EXPROGM (EXPROGM )
-------------------------------------------------------------------------------
                    TEST-CASE                SAMPLE   2          1049     08/05/75
-------------------------------------------------------------------------------
(   1)     EXMPLI  (EXPROGM )
-------------------------------------------------------------------------------
                    TEST-CASE                SAMPLE   3          1049     08/05/75
-------------------------------------------------------------------------------
(   1)     EXMPLI  (EXPROGM )
-------------------------------------------------------------------------------
                    TEST-CASE                SAMPLE   4          1049     08/05/75
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
                              TEST-SET TERMINATION
-------------------------------------------------------------------------------
```

Sample Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODTRACE.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODTRACE.
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER.
END.
```

ANALYZER,MODULE = <name>...

ANALYZER,MODULE = <name>...

Description

The module specification commands are used to identify the modules on LIBOLD for which ANALYZER processes the summary data on the Test Execution trace file. The default is no modules specified. To specify all known modules use the command:

```
ANALYZER,ALL MODULES.
```

To specify a subset of modules use the command:

```
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.
```

The following ANALYZER options require module specification: SUMMARY, HIT, NOTHIT, TIME, ALL.

Rule

As many as 24 modules can be specified using this command.

Example Command Sets

(1)
```
OLD LIBRARY = REFTEST.
START.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,ALL.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,FACTOR = 20.
ANALYZER,TIME.
ANALYZER,MODULE = EXMPL1.
ANALYZER.
END.
```

ANALYZER,NOTHIT.

Description

This command causes ANALYZER to list any DD-path which was not traversed in each test case and all test cases for the specified modules. If a module was not invoked during any test cases, not probed during INSTRUMENT, or not specified for ANALYZER processing, it will not be referenced in this report.

Sample Output

```
JAVS REPORT FOR DD-PATHS NOT EXECUTED.     4 CASE(S)                    8/04/75   2.53.16

-------------------------------------------------------------------------------------------------------------------
MODULE    JAVSTEXT I          TEST              I  PATHS  I          LIST OF DECISION-TO-DECISION PATHS NOT EXECUTED
 NAME       NAME   I      IDENTIFICATION        I NOT HIT I
-------------------------------------------------------------------------------------------------------------------
EXPROGM   EXPROGM  I SAMPLE    1  2209 08/04/75 I     4   I       2   3   4   5
                   I SAMPLE    2  2209 08/04/75 I     3   I       1   3   4
                   I SAMPLE    3  2209 08/04/75 I     3   I       1   3   4
                   I SAMPLE    4  2209 08/04/75 I     4   I       1   2   4   5
                   I    TOTAL NOT HIT           I     1   I       4
-------------------------------------------------------------------------------------------------------------------
EXMPL1    EXPROGM  I SAMPLE    1  2209 08/04/75 I  A L L  I
                   I SAMPLE    2  2209 08/04/75 I    11   I       2   5   6   8   9  10  11  12  14  17  19
                   I SAMPLE    3  2209 08/04/75 I    11   I       2   5   6   8   9  10  11  12  14  17  19
                   I SAMPLE    4  2209 08/04/75 I  A L L  I
                   I    TOTAL NOT HIT           I    11   I       2   5   6   8   9  10  11  12  14  17  19
-------------------------------------------------------------------------------------------------------------------
```

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,NOTHIT.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
FOR LIBRARY.
PRINT,DDP.
END FOR.
END.
```

This command set will provide the post-test analysis coverage summary, the list of DD-paths not executed in any of the test cases, and the list of statements on each DD-path for all modules in the library.

ANALYZER,SUMMARY.

Description

When this option is specified, a report is produced which summarizes the testing coverage of each of the specified modules for each test case encountered on the saved trace file and for the cumulative of all tests completed in the run. Included in the report is the list of modules selected but not invoked during any of the tests in the run. The cumulative coverage shown on the last line is adjusted to reflect all selected modules.

Sample Output Excerpt

DAYS REPORT FOR DD-PATH SUMMARY. 17 CASE(S) 09/29/76

ALL SPECIFIED MODULES ENCOUNTERED ON THE AUDIT FILE 09/29/76

DATA9 1 09/29/76			SUMMARY -- THIS TEST			CUMULATIVE SUMMARY			
MODULE NAME	SYSTEM NAME	NUMBER OF DD-PATHS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE	NUMBER OF TESTS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE
JBPAD	JBPAD	121	1	4	3	1	1	4	3
JBPAD	JBPAD	7	0	0	0	1	0	0	0
JBPAD	JBPAD	7	1	6	85	1	1	6	85
JBPAD	JBPAD	3	3	2	66	1	3	2	66
JBPID	JBPID	61	0	0	0	1	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
RRPT	JBPID	7	0	0	0	17	1	7	100
RRTOB	JBPID	7	0	0	0	17	85	7	100
COMTER	JBPID	7	0	0	0	17	1154	6	85
***** ALL *****		1164		1	0			486	41

ALL SPECIFIED MODULES NOT INVOKED DURING RUN

MODULE NAME	SYSTEM NAME	NUMBER OF DD-PATHS
BCCT	JBPID	5
BCRR	JBPID	1
BCNF	JBPID	1
BTPK	JBPID	1
RSCC	JBPID	3
RSCIX	JBPID	1
***** ALL *****		
UNINVOKED MODULES		12

SUMMARY FOR ALL SPECIFIED MODULES

NUMBER OF DD-PATHS	DD-PATHS TRAVERSED	PER CENT COVERAGE
1176	486	41

ANALYZER,SUMMARY. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,SUMMARY.
ANALYZER.
END.
```

ANALYZER,TIME.

Description

This command causes ANALYZER to print a summary of the time spent in execution within the specified modules for all tests. Only complete invocation-return cycles are included in the time reported (i.e., the increment of time added for each cycle to the cumulative execution time for a module is the time interval logged by the system clock between the invocation of the module and the next return from the module, excluding time spent in the data collection routines).

The TIME report lists all specified modules, even though some modules may not have been invoked or instrumented (denoted by NOT TRACED).

Sample Output

JAVS EXECUTION TIME SUMMARY OUTPUT

MODULE NAME	JAVSTEXT NAME	EXECUTION TIME(MS)	ESTIMATED TIME(MS)	CYCLES TRACED	INVOCATIONS
EXCOMPL	EXCOMPL				NOT TRACED
EXPROGM	EXPROGM	171	143	1	1
EXMPL2	EXPROGM				NOT TRACED
EXMPL1	EXPROGM	26	22	2	2
EXMPL3	EXPROGM				NOT TRACED

In this sample output, the estimated time reflects an ANALYZER, FACTOR = 20. command accompanying the TIME report command. In this case 20% of the actual execution time is subtracted from the values listed under execution time and these reduced values are listed as the estimated execution time. The execution time spent in EXPROGM (171 ms) includes the 26 ms spent in EXMPL1, since EXMPL1 is nested within EXPROGM.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,FACTOR = 20.
ANALYZER,TIME.
ANALYZER,SUMMARY.
ANALYZER.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,TIME.
ANALYZER,CASES = 75.
ANALYZER.
END.
```

In the second command set, no attempt was made to estimate the time spent executing the probe invocation statements; thus the execution and estimated time value will be identical.

ASSIST,CROSSREF,JAVSTEXT = <name>,...

ASSIST,CROSSREF,JAVSTEXT = <name>,...

Description

This command produces a cross-reference listing of items, files, switch names, labels and subprogram names for all the specified JAVSTEXTs. The form of the command is:

ASSIST,CROSSREF,JAVSTEXT = <name-1>,<name-2>,...,<name-n>.

The output for this command is similar to that for the library cross reference, except that only symbols from the specified JAVSTEXTs are included.

Rules

(1) A maximum of 22 JAVSTEXTs can be specified with this command.

(2) A maximum of 125 modules can be used for cross reference mapping.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,JAVSTEXT = EXCOMPL,EXPROGM.
END.
```

ASSIST,CROSSREF,LIBRARY.

Description

The outcome of this command is a cross-reference listing of names in the entire library and their usage. The names listed are all items, files, switch names, labels, and subprogram names.

Rules

(1) A maximum of 100 JAVSTEXTs can be used in cross reference mapping.

(2) A maximum of 125 modules can be used in cross reference mapping.

Sample Output Excerpt

```
GENERAL CROSS REFERENCE LISTING
    FOR WHOLE LIBRARY

SYMBOL    MODULE    USED/SET/DEFINITION ( * INDICATES SET, D INDICATES DEFINITION )

BG        EXPROGM    18D   28
CARD      EXPROGM    12D   18*   21    22    23
EXMPL1    EXMPL1      1
          EXPROGM    24
EXMPL2    EXMPL2      1
          EXPROGM    27
EXMPL3    EXMPL1     29
          EXMPL3      1
FILL      EXMPL1      5D   15*
ID        EXPROGM     5D   15    21
INDXS     EXMPL1      7D   22*   24*   28
ITER1A    EXPROGM     8D   10    15    22
ITER1     EXMPL2      4*
          EXPROGM     6D   10D   24    26
ITER2A    EXPROGM     9D   11    15    23
ITER2     EXMPL2      5*
          EXPROGM     7D   11D   24
LABEL1    EXMPL1     27    27    27    30D
LABEL2    EXMPL1     27    29D
LIMIT1    EXMPL1      1     3D    9    10*   12
LIMIT2    EXMPL1      1     4D   18
MESAGE    EXCOMPL     5D
MESSAG    EXCOMPL    11D
          EXMPL1     31*   33*   35*   38
          EXPROGM    16*   17
MSG1      EXCOMPL     7D
          EXPROGM    16
MSG2      EXCOMPL     8D
          EXMPL1     31
MSG3      EXCOMPL     9D
          EXMPL1     33
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,LIBRARY.
END.
```

Description

There are two forms of this command:

ASSIST,PICTURE. or

ASSIST,PICTURE,<option-1>,<option-2>.

where option-1 is CONTROL and option-2 is NOSWITCH.

This command produces, for the specified module, a stylized picture of the pattern of program flow implicit in the set of sequence of DD-paths. The available options are CONTROL and NOSWITCH. If CONTROL is present, the picture includes all control statements (e.g., GOTO, invocation) on the DD-path. If NOSWITCH is present, the picture omits the flows emanating from GOTO-switch statements.

Each line in the picture represents a statement at the beginning of a DD-path or a control statement; the statement type* and number are displayed in the center. Each DD-path is represented by either a B → E sequence or an S. The B → E sequence shows where the DD-path Begins and Ends; the S indicates that the DD-path begins and ends on the same statement. The statements shown are those which are "necessary" to display all possible program flow and, by convention, are arranged in statement order. All "forward flow" (with respect to the statement ordering) is shown by convention on the right-hand side of the picture; and all "reverse flow" is shown in the left-hand side. For those statements which are at the beginning of a DD-path, the number of the DD-path is specified on the far right margin. The DD-path numbers are ordered left to right. Control statements are indicated by a C.

For very complex modules (e.g., switches with a large number of targets) the NOSWITCH format limits the number of paths which can be displayed. If the picture overflows on the right or left side, an informative message is printed and processing continues.

*See Appendix C for statement type abbreviations.

ASSIST,PICTURE. (Cont.)

Sample Output

```
PICTURE WITH ALL DD-PATHS...
MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

(B = BEGIN, E = END, S = SELF-LOOP)          STMT TYPE   STMT NO.      DD-PATH NUMBERS...
                                             <PROC    1> B                1
                                                         +
                                             <IF      9> EBB              2  3
                                                         ++
                                          EE <IFEI   21> BEEB             4  5
                                          ++             + +
                                          ++ <ORIF   23> +B E             6
                                          ++             ++
                                          ++ <BTSW   28> EEBBBBB          7  8  9  10  11
                                          ++               +++++
                                             <BTCL   29>                                  C
                                          ++ <IFEI   30> BBEEEEE
                                          ++             ++
                                          ++ <ORIF   32> +EBB             14 15
                                          ++             + ++
                                          ++ <ORIF   34> +B+E             16
                                          ++             +++
                                          +B <END    39> EEEB             18 17
                                          +                 +
                                          B  <END    41> B  E             20 19
                                                         +
                                             <END    44> E
```

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
MODULE = EXMPL1.
ASSIST,PICTURE,CONTROL.
END.
```

This command set produced the sample output.

(2)
```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
ASSIST,PICTURE.
END FOR.
END.
```

This command set would produce a picture for each module on the library similar to the one in the sample output, but without the control statements (GTCL 29 in the sample output) displayed.

(3)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
ASSIST,PICTURE,CONTROL,NOSWITCH.
END FOR.
END.
```

In addition to the control statements' being displayed, this command set will omit the flows and DD-path numbers emanating from GOTO-switch statements (GTSW 28, DD-paths 7 to 11 in the sample output).

(4)
```
OLD LIBRARY = REFMAN.
START.
MODULE = EXMPL1.
ASSIST,PICTURE,NOSWITCH.
END.
```

The picture options need not be in order.

ASSIST,REACHING SET,<specs>.

Description

This command is used to analyze the flow required to reach a particular DD-path according to the items in the specification list. The reaching set consists of all DD-paths which flow between the beginning and ending DD-paths. There are three types of specifications (which may be present in any order but separated by commas):

<number-to>{,<number-from>}

ITERATIVE

PICTURE

The DD-path for the reaching set (i.e., the target of flow) is named by the (required) specification <number-to>. In the absence of a <number-from> specification, the flow is assumed to start with the first executable statement. The <number-from> specification allows the user to designate the DD-path where flow starts. The analysis begins with the first executable statement on the DD-path.

The (optional) ITERATIVE specification allows the user to control the set of DD-paths in the analysis. If ITERATIVE is not specified, all flows which include iteration are suppressed in determination of paths of control. If ITERATIVE is specified, the flows include iteration.

The (optional) PICTURE specification allows the user to control the additional reports presented. If PICTURE is not present, the output is limited to the reaching set definition. The additional picture report shown in the sample output is included as a result of the command

ASSIST,REACHING SET,17,4,PICTURE.

Rules

(1) Maximum of 100 DD-paths per reaching set path.

(2) Maximum of 100 outways per decision.

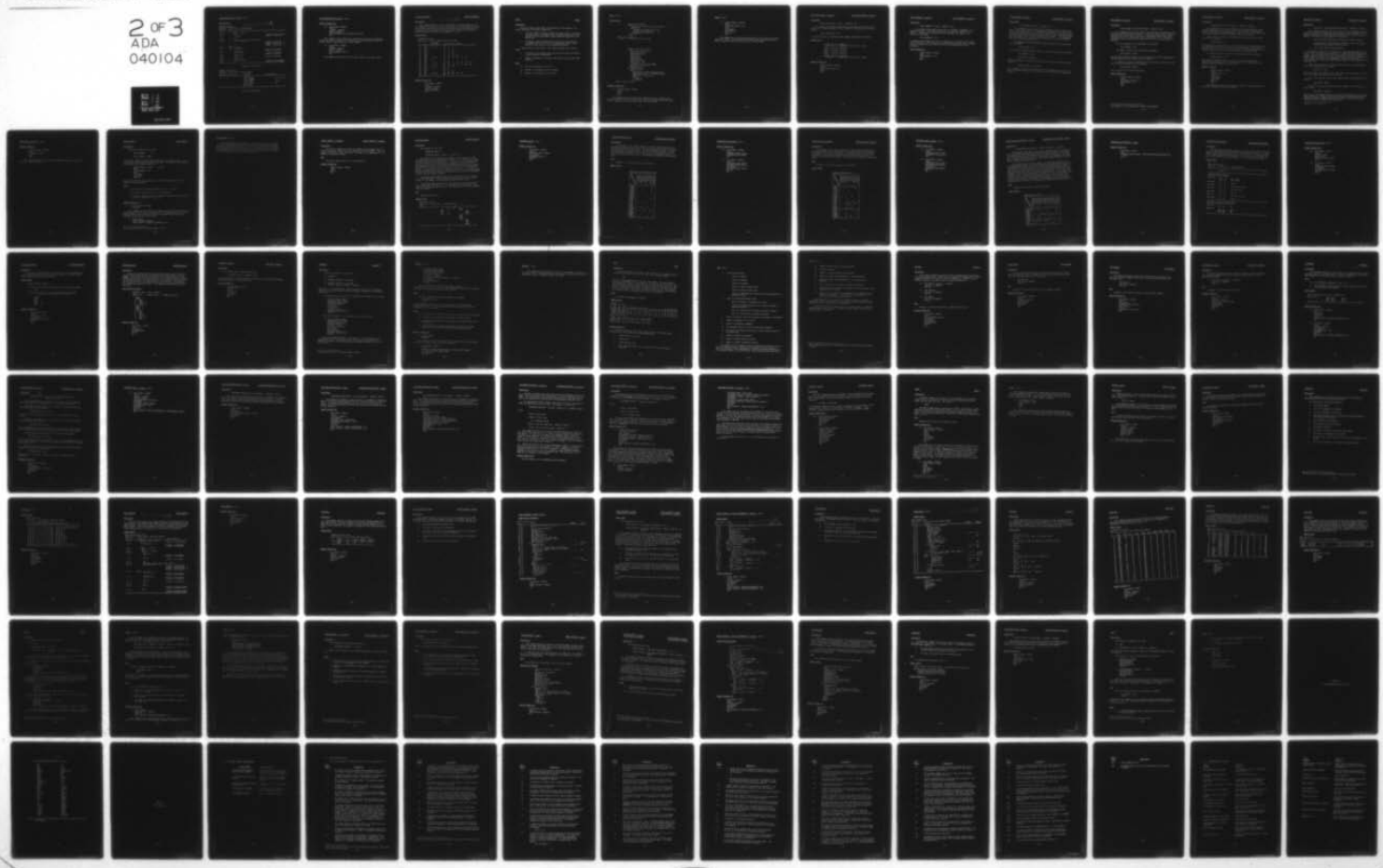
(3) Maximum of 1200 DD-paths per analyzed module for reaching set.

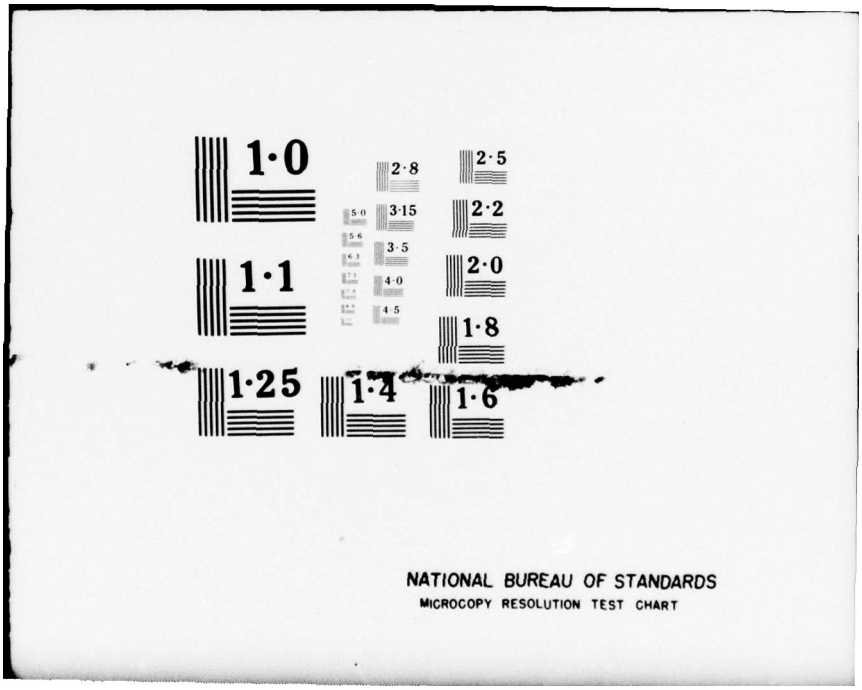
(4) Maximum of 2400 statements per analyzed module for reaching set.

(5) Maximum of 200 statements in reaching set.

AD-A040 104 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JAVS TECHNICAL REPORT. REFERENCE MANUAL.(U)
APR 77 C GANNON, N B BROOKS F30602-76-C-0233
UNCLASSIFIED RADC-TR-77-126-VOL-2 NL

2 OF 3
ADA
040104




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

ASSIST,REACHING SET,<specs>. (Cont.)

Sample Output


```
NON-ITERATIVE REACHING SET
   OF DD-PATH  17 FROM DD-PATH   4

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

------------------------------------------------------------------------------------------------------------
                  * * *
 18 ( 2)                 FOR J = 1 , 1 , LIMIT2 $
 19 ( 3)                   BEGIN
                  * * *
 21 ( 3)                   IFEITH J LQ 3 $                            *ESSENTIAL PREDICATE*  BEGINNING DDP
                                                                     ** DD-PATH   4 IS TRUE BRANCH
 22 ( 4)                     INDXS = J $
                  * * *
 25 ( 3)                   END
                  * * *
 27 ( 3)                   SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
 28 ( 3)                   GOTO PICK ($ INDXS - 1 $) $
                                                                     ** DD-PATH   7 IS SWITCH OUTWAY    1
                                                                     ** DD-PATH   8 IS SWITCH OUTWAY    2
                                                                     ** DD-PATH   9 IS SWITCH OUTWAY    3
                                                                     ** DD-PATH  10 IS SWITCH OUTWAY    4
                                                                     ** DD-PATH  11 IS SWITCH OUTWAY    5
 29 ( 3)          LABEL2.
                           GOTO EXMPL3 $
 30 ( 3)          LABEL1.
                           IFEITH RESULT LS 4 $
                                                                     ** DD-PATH  12 IS TRUE BRANCH
                                                                     ** DD-PATH  13 IS FALSE BRANCH
 31 ( 4)                     MESSAG = MSG2 $
 32 ( 3)                   ORIF RESULT EQ 4 $
                                                                     ** DD-PATH  14 IS TRUE BRANCH
                                                                     ** DD-PATH  15 IS FALSE BRANCH
 33 ( 4)                     MESSAG = MSG3 $
 34 ( 3)                   ORIF 1 $
                                                                     ** DD-PATH  16 IS TRUE BRANCH
 35 ( 4)                     MESSAG = MSG4 $
 36 ( 3)                   END
                  * * *
 38 ( 3)                   OUTPUT PRINTR MESSAG $                     *ESSENTIAL PREDICATE*  ENDING DDP
 39 ( 3)                   END                                       ** DD-PATH  17 IS LOOP ON FOR AGAIN
------------------------------------------------------------------------------------------------------------
```

Reaching Set Definition

```
PICTURE WITH A SET OF DD-PATHS...

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

                                            STMT  STMT
(B = BEGIN, E = END, S = SELF-LOOP)         TYPE   NO.                          DD-PATH NUMBERS...
------------------------------------------------------------------------------------------------------------
                                          E <IFEI  21> B                                 4
                                          *            *
                                          * <GTSW  28> EBBBBB                            7   8   9  10  11
                                          *             *****
                                          * <IFEI  30> BEEEEEB                           12  13
                                          *            *     *
                                          * <ORIF  32> *BB   E                           14  15
                                          *            ***
                                          * <ORIF  34> **EB                              16
                                          *            ** *
                                          B <END   39> EE E                              17
------------------------------------------------------------------------------------------------------------
```

Picture of Reaching Set

ASSIST,REACHING SET,<specs>. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST,REACHING SET,16,ITERATIVE,PICTURE.
END.
```

This command set will produce the reaching set definition and reaching set picture reports containing all DD-paths, including iteration, which flow between DD-path 1 and DD-path 16 of module EXMPL1.

(2)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST,REACHING SET,17,4.
END.
```

This command set produces the first report shown in the sample output.

Description

This command produces a report describing the DD-path membership for each executable statement in the "current module." The statements are listed in statement order with their type, the DD-path numbers which begin on the statement, and the complete set of DD-paths which contain the statement. A DD-path beginning or ending is represented by a B or E.

Sample Output

STATEMENT/DDPATH LISTING

MODULE <EXMPL1 >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

STMT	TYPE	DD-PATHS BEGUN BY STATEMENT	DD-PATHS CONTAINING STATEMENT
1	PROC	(1)	1B
2	BEGN		1
9	IF	(2- 3)	1E 2B 3B
10	ASMT		2
11	ASMT		2 3
12	FOR3		2 3 19
13	BEGN		2 3 19
15	ASMT		2 3 19
16	ASMT		2 3 19
18	FOR3		2 3 17 19
19	BEGN		2 3 17 19
21	IFEI	(4- 5)	2E 3E 4B 5B 17E 19E
22	ASMT		4
23	ORIF	(6)	5E 6B
24	ASMT		6
25	END		4 6
28	GTSW	(7- 11)	4E 6E 7B 8B 9B 10B 11B
29	GTOL		10 11
30	IFEI	(12- 13)	7E 8E 9E 10E 11E 12B 13B
31	ASMT		12
32	ORIF	(14- 15)	13E 14B 15B
33	ASMT		14
34	ORIF	(16)	15E 16B
35	ASMT		16
36	END		12 14 16
38	OUTP		12 14 16
39	END	(17- 18)	12E 14E 16E 17B 18B
41	END	(19- 20)	18E 19B 20B
44	END		20E

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST,STATEMENTS.
END.
```

Description

The BASIC command causes BASIC syntax analysis to be executed. The actions performed by the BASIC command are:

1. To build LIBNEW containing a module descriptor block, a statement descriptor table, a statement table, a symbol locator table, and a symbol table for each module in each START-TERM sequence on the INPUT file.

2. To produce a report indicating the source text of each module, the module name, and a count of the syntax and semantic errors detected in the source for each START-TERM sequence.

A sample report is shown below. This report contains the following items:

- A listing of the original source text with the errors encountered in BASIC analysis interspersed.

- Summary information at the end of each module and each START-TERM sequence.

Rules

(1) See BASIC constraints in Sec. 3.2.

(2) Maximum of 250 modules can be on LIBNEW.

(3) Maximum of 150 modules in any JAVSTEXT.

BASIC. (Cont.)

Sample Output

```
                    ##,JAVSTEXT EXCOMPL PRESET##
                    START$
                         ##COMPOOL EXAMPLE CONTAINING PRESET OUTPUT MESSAGES  ##
                         COMMON MESAGE$
                         BEGIN
                              ITEM MSG1 H 18 P 18H(1JAVS TEST CASE    )$
                              ITEM MSG2 H 18 P 18H(0RESULT LT 4       )$
                              ITEM MSG3 H 18 P 18H(0RESULT EQ 4       )$
                              ITEM MSG4 H 18 P 18H(0RESULT GT 4       )$
                         ITEM MESSAG H 18$
                         END
                     TERM$

EXCOMPL  (EXCOMPL ) COMPLETED
    ********     NO ERRORS WERE FOUND BY JAVS-2  ******
```

```
                    ##,JAVSTEXT EXPROGM COMPUTE (EXCOMPL)##
                     START$
                         ##JOVIAL SIMPLE TEST PROGRAM ##
                         DEFINE INTG ##I 24 S ##$
                         DEFINE HLL ## H 4 ##$
                         DEFINE NBYTWO ## 4 ##$
                         ITEM ID HLL$
                         ITEM ITER1 INTG$
                         ITEM ITER2 INTG$
                         ITEM ITER1A HLL$
                         ITEM ITER2A HLL$
                         OVERLAY ITER1 = ITER1A$
                         OVERLAY ITER2 = ITER2A$
                         ITEM CARD H 80$
                         FILE READER H 0 R 84 V(OK) V(EOF) TAPE5$
                         FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6$
                         MONITOR ID, ITER1A, ITER2A$
                         MESSAG = MSG1$
                         OUTPUT PRINTR MESSAG$
                     BG. INPUT READER CARD$
                         IF READER NQ V(EOF)$
                         BEGIN
                              BYTE($0,NBYTWO$) (ID) = BYTE($0,NBYTWO$) (CARD)$
                              BYTE($0,NBYTWO$) (ITER1A)= BYTE($9,NBYTWO$) (CARD)$
                              BYTE($0,NBYTWO$) (ITER2A)= BYTE($19,NBYTWO$) (CARD)$
                              EXMPL1(ITER1,ITER2)$
                              CLOSE EXMPL2$      ## MAIN CLOSE ##
                              BEGIN
                                   ITER1 = 1$
                                   ITER2 = 1$
                              END ##EXMPL2##

EXMPL2   (EXPROGM ) COMPLETED
```

Example Command Sets

(1)
```
CREATE LIBRARY = REFMAN.
START.
BASIC.
END.
```

This command set will cause library REFMAN to be built containing the syntax analysis for the input source text. All BASIC default options are used in this command set, and the sample output will be produced.

(2)
```
CREATE LIBRARY = REFMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC.
FOR LIBRARY.
PRINT,MODULE.
END FOR.
END.
```

This command set will perform syntax analysis on the input source text, create the library, print an abbreviated report, and then print the reformatted listing for each module on the newly created library.

BASIC,CARD IMAGES = <option>.

BASIC,CARD IMAGES = <option>.

Description

BASIC,CARD IMAGES = ON/OFF. (DEFAULT = ON.)

This command allows user control over the inclusion of the source text listing in the report produced by BASIC processing. The effect of the option command

BASIC,CARD IMAGES = OFF.

on the BASIC output is shown below; only summary information is provided.

Sample Output

```
EXCOMPL  (EXCOMPL ) COMPLETED
    ********    NO ERRORS WERE FOUND BY JAVS-2  ******

EXMPL2   (EXPROGM ) COMPLETED

EXMPL3   (EXPROGM ) COMPLETED

EXMPL1   (EXPROGM ) COMPLETED

EXPROGM  (EXPROGM ) COMPLETED
    ********    NO ERRORS WERE FOUND BY JAVS-2  ******
```

Example Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC.
END.
```

BASIC,COMMENTS = <option>. BASIC,COMMENTS = <option>.

Description

BASIC,COMMENTS = ON/OFF. (DEFAULT = ON.)

This command allows user control over the treatment of comments in the module source text. The default action is to include all comments in the library. Comments may be excluded from the statement text table of all modules by the command

BASIC,COMMENTS = OFF.

If comments are included, more space is required on the library and all subsequent processing of the statement table requires more computer time. Comments should be excluded from the library if they are imbedded within JOVIAL statements.

Sample Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,COMMENTS = OFF.
BASIC.
END.
```

BASIC,DEFINES = <option>.

Description

BASIC,DEFINES = ON/OFF. (DEFAULT = ON.)

This command allows the user control over BASIC usage of DEFINE variables. The default option is to replace the occurrence of each DEFINE variable with its definition just as the JOVIAL compiler does. Subsequent BASIC processing utilizes the results of that definition both in statement analysis and in the text stored for each statement on the library. It is essential that the default option be used if the modules are to be processed by any other JAVS standard processing step.

Replacement of DEFINE variables by their definition may be suppressed with the command:

BASIC,DEFINES = OFF.

This command is normally used only when the following commands are also used:

BASIC,ERRORS = OFF.

BASIC,TEXT = JAVSTEXT.

Text so processed may be written to LPUNCH in the structured format with the command:

PUNCH,JAVSTEXT = <text-name>.

This is useful in using JAVS to punch a copy of reformatted source text. To get a reformatted listing of the source text, the PRINT,JAVSTEXT command can be used.

BASIC,ERRORS = <option>.

Description

BASIC,ERRORS = ON/OFF/LIMIT/TRACE. (DEFAULT = ON.)

This command allows the user control over the syntax and semantic errors detected in the JOVIAL source by BASIC. The default option prints a numbered error message as each error is detected.* These error messages are interspersed in the list of source text lines (CARD IMAGES = ON.). Since BASIC processing scans the JOVIAL source text only once, not all source errors are detected.

All error messages may be suppressed by the command

BASIC,ERRORS = OFF.

The number of errors may be limited by the command

BASIC,ERRORS = LIMIT.

For this option, when the number of errors exceeds 100, BASIC terminates processing immediately and a fatal JAVS error results.

In addition to the error message itself, the origin of the error in the JAVS code can be displayed with the command

BASIC,ERRORS = TRACE.

This is useful in JAVS system maintenance.

Sample Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,CARD IMAGES = OFF.
BASIC,ERRORS = OFF.
BASIC.
END.
```

*See Appendix B for a description of BASIC error messages.

BASIC,SYMBOLS = <option>.

Description

BASIC,SYMBOLS = ON/OFF/PARTIAL. (DEFAULT = PARTIAL.)

This command allows the user control over the symbols entered in the detailed symbol table. The default action is to enter only those symbols essential to other JAVS processing steps. These symbols, called control symbols, are the names of modules, statement labels, and switches.

Other JOVIAL identifiers and constants may be included in the symbol table with the command

BASIC,SYMBOLS = ON.

BASIC performs a limited analysis on the properties of each symbol and stores type-dependent information about each in the symbol table. More space is required on the library for the additional symbol table entries and all subsequent processing of the symbol table requires more computer time.

The symbol table may be omitted from the library with the command

BASIC,SYMBOLS = OFF.

This option is meaningful only if the library is never processed by STRUCTURAL and subsequent JAVS processing steps (e.g., when just BASIC processing is used to reformat the source text).

Sample Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,SYMBOLS = ON.
BASIC.
FOR LIBRARY.
PRINT,STB.
END FOR.
END.
```

This command set will put all symbols on the newly created library and then print the symbol table for each module.

BASIC,TEXT = <option>.

Description

BASIC,TEXT = COMPUTE/BOTH/PRESET/JAVSTEXT. (DEFAULT = COMPUTE.)

This command allows the user control over the type of analysis performed by BASIC on the sets of START-TERM text appearing on the file INPUT. BASIC processes a sequence of START-TERM texts until an end-of-file is detected on READER. Three types of text sequences are permitted:

1. One or more START-TERM texts of executable (COMPUTE) source.

2. A single COMPOOL START-TERM text followed by one or more START-TERM texts of executable (COMPUTE) source.

3. One or more COMPOOL START-TERM texts.

The default action is to process type 1. The text presented under the COMPUTE option is assumed to be a sequence of START-TERM texts which are executable. Any particular START-TERM text may normally require a COMPOOL for proper compilation; BASIC will properly process the text in the absence of its COMPOOL under the COMPUTE option provided a JAVSTEXT directive (Sec. 1.4) which includes the related-text-name parameter appears at the beginning of the START-TERM text.

Type 2 text sequences (one COMPOOL followed by one or more executable texts) are processed with the command:

BASIC,TEXT = BOTH.

With this option, when LIBOLD is used, the COMPOOL must be included on READER,* even if no change has been made to its source.

Type 3 text sequences (one or more COMPOOL texts) are processed with the command:

BASIC,TEXT = PRESET.

BASIC will not separate source text into modules if the following command is used:

BASIC,TEXT = JAVSTEXT.

Since the unit of processing for most other JAVS processing steps is an executable module, this command prohibits performing module analyses on the text. (See BASIC,DEFINES = command description for usage.) All errors normally detected by BASIC are ignored with this option.

---

*See Table 1.1 on page 1-4.

BASIC,TEXT = <option>. (Cont.)

Example Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,TEXT = BOTH.
BASIC.
END.
```

This command set will build a library containing a single COMPOOL and one or more START-TERM texts.

Description

This macro command has two forms:

```
BUILD LIBRARY.        or

BUILD LIBRARY = <name>.
```

In the first command form, the library name given to LIBNEW is TEST. In the second form, the user specifies the library name. Both command forms generate the following JAVS standard commands:

```
CREATE LIBRARY = <name>.     (or TEST)
START.
BASIC,COMMANDS = OFF.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

The macro command processor generates the JAVS "END" command if it is not present as the last command.

Rules

(1) See BASIC and STRUCTURAL constraints in Sec. 3.2 and 3.3.

(2) The library name should be 8 or less characters.

(3) This macro command should not be used if the source program includes a COMPOOL (constraint 2 on page 3-3).

Example Command Sets

(1)
```
BUILD LIBRARY = REFMAN.
DOCUMENT.
```

This command set will create a new library, perform syntax and structural analyses and produce a series of JAVS reports for all modules in the JOVIAL source program. These reports are useful for documentation, maintenance, testing and retesting the source program.

(2)
```
BUILD LIBRARY.
PROBE,JAVSTEXT = EXPROGM.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
```

---

*Can be used only with the overlay version of JAVS.

BUILD LIBRARY. (Cont.)

This command set shows a mixture of JAVS macro and standard commands. The first macro command will create a new library called TEST and perform syntax and structural analyses on all modules in the JOVIAL source program. The PROBE macro command will instrument JAVSTEXT EXPROGM, using the default instrumentation options, and write the instrumented text onto file LPUNCH. The last command will print the instrumented text.

CREATE LIBRARY = <libname>. CREATE LIBRARY = <libname>.

Description

This library command specifies that LIBNEW (a "new library") will be created during the current run, as is normally the case in BASIC. When this option is specified, LIBNEW is assumed to be empty at the beginning of a run. LIBNEW is a read/write library.

Rule

The library name should be 8 or less characters.

Example Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC.
END.
```

Description

This command has two forms:

DEPENDENCE,BANDS. or

DEPENDENCE,BANDS = <number>. (DEFAULT = 2.)

The outcome of this command is a "snapshot" of the position of the selected module within the inter-module hierarchy. The sample output shows the BANDS report for the default width. To the left of the selected module is shown the structure of the calls to the module; to the right of the selected module is shown the invocation structure emanating from the module. The number of bands is the width (in each direction) of the structure displayed. Up to five bands may be displayed on this report. This report is useful in determining the extent of inter-module dependence to several levels. Modules which are called from only one other module are potential candidates to head a link for overlay purposes.

The modules listed under column -1 call the selected module directly (i.e., GTCR and OLOOK). The modules listed under column -2 call those in column -1. For example, GTCR is called by CKBK, LUK, and OLOOK.

The modules listed under column 1 are called by the selected module (e.g., BTOH, ERRXR, GTCARD, etc.). The modules listed under column 2 are called by those listed under column 1. For example, ERRXR calls BTOD, ERROR, OPUT, and SYSTEMP.

Rule

Maximum bandwidth is 5.

Sample Output

```
MODULE INVOCATION BANDS

MODULE <NPUT    >, JAVSTEXT <NPUT    >, PARENT MODULE <NPUT    >

LEVEL                          -2        -1        0        1        2
-------------------------------------------------------------------------------
                                                  NPUT
                                          GTCR             BTOH
                                CKBK                       ERRXR
                                LUK                                 BTOD
                                OLOOK                               ERROR
                                                  OLOOK             OPUT
                                LOOK                                SYSTEMP
                                                           GTCARD
                                                           HOLS
                                                           INMOB
                                                           PRXX
                                                           TERMIN
                                                                    BTOD
                                                                    ERRXR
                                                                    FATAL
                                                                    OPUT
                                                                    WRAPUP
-------------------------------------------------------------------------------
```

DEPENDENCE,BANDS. (Cont.)

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
FOR LIBRARY.
DEPENDENCE,BANDS = 5.
END FOR.
END.
```

DEPENDENCE,GROUP,AUXLIB.

Description

The outcome of this library command is a report which identifies all modules not present in the library which are called by modules in the library. The sample output shows this report. The data presented is useful in analyzing interfaces of all the modules on the library to other software. Each X mark represents an invocation by the module in the list down the page to the module in the list across the page. The modules are presented in alphabetical order.

Rule

Maximum of 100 library modules for this report.

Sample Output

AUXILIARY LIBRARY DEPENDENCE TABLE

```
+---------+-------------------------+
+.I       +                         +
+I.N      +ABEFGGGHIIMMMMOPPPRHSS#+
+ N.V     +CIRAETVONNDDDUPRUUETDYR+
+  V.O    +SURTTCELMSARBVUXTTMSBSA+
+   O.K   +THOARAUSDHNNS2TXHLQ STP+
+    K.E  +B RLLRP R AET   LSU TEU+
+     E.E+    KD    MX    KTO RMP+
+      R.+                    P  +
+---------+-------------------------+
+ ADCR    +                         +
+ ADDTOK  +       X           XX    +
+ BTOD    +                         +
+ CIONT   +                     X   +
+ CKRK    +                         +
+ CONTOK  +       X             X   +
+ DTOB    +                         +
+ EFAC    +                         +
+ ERRXR   + X             X      X  +
+ GETB    +                     X   +
+ GETC    +                     X   +
+ GTCR    +                         +
+ LOOK    +       X  X              +
+ LUK     +             X           +
+ MUV1    +                         +
+ MUV4    +                         +
+ NPUT    + X   X XX      X         +
+ OLOOK   +                         +
+ OUTSB   +X   X    X XX   XX  X    +
+ PRM     +                         +
+ PVALU   +      X            X     +
+ SBPTR   +                         +
+ SGNP    +                         +
+ TERMIN  +  X           X        X+
+---------+-------------------------+
```

DEPENDENCE,GROUP,AUXLIB. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
END FOR.
END.
```

DEPENDENCE,GROUP,LIBRARY.

Description

The outcome of this library command is a report which identifies the interdependencies of all known modules. The sample output shows a sample of this report. Only modules present in the library are listed. Each X mark represents an invocation by the module in the list down the page to the module in the list across the page. The modules are presented in alphabetical order.

Rule

Maximum of 100 library modules for this report.

Sample Output

LIBRARY DEPENDENCE TABLE

```
+--------+------------------------------+
+.I      +                              +
+I.N     +AABCCCDEEGGGLLMMNOOPPSST+
+ N.V    +DDTIKOTFREETOUUUPLURVAGE+
+  V.O   +CDOURNOARTTCOKVVUOTMAPNR+
+   O.K  +RTDNKTRCXRCRK 14TOS LTPM+
+    K.E + O T O  R         KB UR I+
+     E.E+ K   K                   N+
+      R.+                         +
+--------+------------------------------+
+ ADCR   +.       X      XX            +
+ ADDTOK + .   X  X          XX        +
+ BTOD   +  .                          +
+ CIDNT  +   .                         +
+ CKRK   +    .      X                 +
+ CONTOK +     .                       +
+ DTOB   +      .                      +
+ EFAC   +  X    .      XX             +
+ ERRXR  +  X     .                    +
+ GETB   +         .                   +
+ GETC   +          .                  +
+ GTCR   +         X  .    X       X   +
+ LOOK   + X       X   .     XXX       +
+ LUK    +   X     X XX .X      X      +
+ MUV1   +         X      .            +
+ MUV4   +          X    X.           +
+ NPUT   +         X        .      X   +
+ OLOOK  +X XXX XXXX X XXXX.  X XX+
+ OUTSB  +                    .        +
+ PRM    +                   . X       +
+ PVALU  +         X           .       +
+ SBPTR  +         X            .      +
+ SGNP   +         X    XX       .     +
+ TERMIN +  X      X              .     +
+--------+------------------------------+
```

DEPENDENCE,GROUP,LIBRARY. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
END FOR.
END.
```

DEPENDENCE,GROUP,MODULES = <name>.

DEPENDENCE,GROUP,MODULES = <name>.

Description

DEPENDENCE,GROUP,MODULES = <name-1>,<name-2>,...,<name-n>.

This library command allows the user to identify a group of modules for which inter-group and intra-group computations are performed on the library. The modules so grouped are related in some way known by the user. For example, the group may be a subset of all the modules in the library which implement a particular functional process. The list of modules named in this command is called the selected group.

The outcome of this library command is a report which identifies all interdependencies between the selected group and the remainder of the library. The sample output shows a sample of the report generated by this command. Each X mark in the matrix represents an invocation by the module in the list down the page to the module in the list across the page. In the upper left quadrant of the matrix are all intra-group dependencies. In the upper right quadrant are all other modules called by members of the group (whether on the library or not). In the lower left quadrant are all other modules in the library which call one or more members of the group. In the lower right quadrant are the direct invocations of other library modules to all other modules called by the group. The modules are presented in alphabetical order.

Rule

Maximum of 115 modules in the invokee groups.

Sample Output

GROUP DEPENDENCY TABLE

```
+---------+--------+-----------------------------------+
+.I       +        +                                   +
+I.N      +ACGGOOPS+AABCCDEEGGHILMMMMNPPRRRSST+
+ N.V     +OOEELURR+CDTIKTFRETONUUDUUPUUVETDGE+
+  V.O    +ONTTOTMP+SCODRUARTCLSKRRVVUTTAMSRNR+
+   O.K   +TTACOS T+TRDNKBCXRRSR NS14TRLLQ SPM+
+    K.E  +OO  KR R+R  T   RL    ET   LSUU T I+
+     E.E+KK       +        K    X    KT O R N+
+      R.+         +                             +
+---------+--------+-----------------------------------+
+ ADDTOK  +.X   XX +       X  X          XX     +
+ CONTOK  + .      +          X          X      +
+ GETB    +  .     +                     X      +
+ GETC    +   .    +                     X      +
+ OLOOK   +  X .   + XXXXXXX X  X  XXX  X    XX+
+ OUTSB   +     .  +X       X  X XX   XX   X   +
+ PRM     +      .X+                           +
+ SBPTR   +       .+       X                    +
+---------+--------+-----------------------------------+
+ LOOK    +X   XXX +       X  X                 +
+ LUK     +  X     +   X   X X     X     X      +
+ MUV4    + X      +               X     X      +
+---------+--------+-----------------------------------+
```

DEPENDENCE,GROUP,MODULES = <name>.

Example Command Set

```
OLD LIBRARY = BIGLIB.
START.
DEPENDENCE,GROUP,MODULES = ADDTOK,PRM,OUTSB,CONTOK,GETB,GETC,
   SBPTR.
END.
```

DEPENDENCE,PRINT,INVOKES.

Description

The outcome of this module command is a report which shows (1) the invocations of the selected module from all other known modules and (2) the invocations within the selected module to all other modules. The sample output shows a report produced by this command. For each module the statement number of the invocation and the source text for the invocation are shown.

Sample Output

```
MODULE INVOCATION SPACE...
MODULE <NPUT    >, JAVSTEXT <NPUT    >, PARENT MODULE <NPUT    >
-------------------------------------------------------------------------------------------------------
  3             PROC NPUT $                                                              (  1)
-------------------------------------------------------------------------------------------------------
INVOCATIONS FROM WITHIN THIS MODULE
-----------------------------------
MODULE ATOH
                    STMT #   68        ATOH ( LYNCNT )
                    STMT #   69        ATOH ( STCTR )
MODULE ERRXR
                    STMT #   29        ERRXR ( 4 )
MODULE GTCARD
                    STMT #   41        GTCARD ( # STATUS , CRD1 )
MODULE HOLS
                    STMT #   18        HOLS ( HDEF ($ A $) )
MODULE INMOB
                    STMT #   46        INMOB ( 0 )
MODULE PRXX
                    STMT #   79        PRXX ( HEL , 120 , EJCT )
MODULE TERMIN
                    STMT #   49        TERMIN ( 1 )
-------------------------------------------------------------------------------------------------------
INVOCATIONS TO THIS MODULE FROM WITHIN LIBRARY
----------------------------------------------
MODULE GTCR
                    STMT #   31        NPUT
MODULE OLOOK
                    STMT #  119        NPUT
                    STMT #  306        NPUT
                    STMT #  363        NPUT
-------------------------------------------------------------------------------------------------------
```

DEPENDENCE,PRINT,INVOKES. (Cont.)

Example Command Sets

(1)
```
OLD LIBRARY = BIGLIB.
START.
JAVSTEXT = NPUT.
MODULE = NPUT.
DEPENDENCE,PRINT,INVOKES.
END.
```

(2)
```
OLD LIBRARY = BIGLIB.
START.
DEPENDENCE,GROUP,LIBRARY.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
END FOR.
END.
```

Description

This library command results in an analysis of the interdependencies of all known modules for interfaces to other software. The sample output shows the report produced by this command.

Sample Output

```
LIBRARY DEPENDENCE SUMMARY...

THE FOLLOWING PROCEDURES ARE NOT INVOKED BY ANY MODULE ON THE LIBRARY

        LOOK

THE FOLLOWING PROCEDURES DO NOT INVOKE ANY MODULE ON THE LIBRARY
        ( * PROCEDURES DO NOT INVOKE ANY MODULES AT ALL )

        CONTOK
        OUTSB
        GETB
        GETC
        BTOD      *
        CIONT
        DTOB      *
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,SUMMARY.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
END FOR.
END.
```

Description

This module command displays the hierarchical structure of all modules invoked from the named module. The sample output shows a report from this command. The level in the hierarchy is shown in the first column. The next column shows all modules on the first level. Each succeeding column indicates the relative position of the named module in the hierarchy. Those modules with an asterisk shown to the right are not on the library. Structures which are repeated are indicated by (ETC.). The modules are alphabetical within each level.

Sample Output (Excerpt)

```
DEPENDENCE TREE--- (*NOT IN LIBRARY)

MODULE <NPUT     >, JAVSTEXT <NPUT     >, PARENT MODULE <NPUT     >

   1 BTOH     *
   1 ERRXR
   2      BTOD
   2      ERROR    *
   2      OPUT     *
   2      SYSTEMP  *
   1 GTCARD   *
   1 HOLS     *
   1 INMDB    *
   1 PRXX     *
   1 TERMIN
   2      BTOD
   2      ERRXR
   2        (ETC.)
   2      FATAL    *
   2      OPUT     *
   2      WRAPUP   *
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
DEPENDENCE,TREE.
END FOR.
END.
```

DESCRIBE = <option>.

DESCRIBE = <option>.

Description

This library option command has the form:

DESCRIBE = ON/OFF. (DEFAULT = OFF.)

This option turns ON or OFF a sequence of printouts which describe the library manager information. The default is OFF.

Example Command Set

```
ALTER LIBRARY = REFMAN.
DESCRIBE = ON.
START.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

Description

This macro command has three forms:

(1) DOCUMENT.

(2) DOCUMENT,JAVSTEXT = <text-name>.

(3) DOCUMENT,JAVSTEXT = <text-name>,
MODULE = <name-1>,<name-2>,...<name-n>.

Each form of the DOCUMENT macro command generates a series of JAVS standard commands which produce reports useful for program documentation, maintenance, and testing.

The expansion of each form of the DOCUMENT macro command is as follows:

(1)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
FOR LIBRARY.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

This form (DOCUMENT.) is for documenting the entire library.

(2)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
JAVSTEXT = <text-name>.
FOR JAVSTEXT.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

This form (DOCUMENT,JAVSTEXT = <text-name>.) is for documenting module interdependencies for the entire library, producing a library-wide cross reference of symbols, and generating module documentary reports for the specified JAVSTEXT.

*Can be used only with the overlay version of JAVS.

DOCUMENT. (Cont.)

(3)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
JAVSTEXT = <text-name>.
FOR MODULE = <name-1>,<name-2>,...,<name-n>.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
END FOR.
```

This form (DOCUMENT,JAVSTEXT = <text-name>, MODULE = ...) produces the same report as in (2), except that the module reports are generated only for the specified modules in the JAVSTEXT.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY, DOCUMENT,PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) A maximum of 100 JAVSTEXTs can be used in a cross reference mapping.

(2) A maximum of 100 modules can be used in the GROUP reports.

(3) A maximum of 23 modules can be specified in the second form of the DOCUMENT macro.

(4) The DOCUMENT macro command requires that syntax and structural analyses have already been performed on the entire library.

Example Command Sets

(1)
```
BUILD LIBRARY.
DOCUMENT.
```

This command set will create a new library called TEST and produce JAVS documentation reports for all modules on the library.

(2)
```
OLD LIBRARY = REFMAN.
START.
DOCUMENT,JAVSTEXT = EXPROGM,MODULE = EXMPL1,EXMPL2,EXMPL3.
FOR MODULE = EXMPL1,EXMPL2,EXMPL3.
PRINT,DDPATHS.
END FOR.
```

DOCUMENT. (Cont.)

This command set uses an existing library called REFMAN to generate the librarywide module interdependency and cross reference reports and the module documentary reports for EXMPL1, EXMPL2, and EXMPL3.

END. END.

Description

After the operations indicated by the contents of the command file are complete, JAVS must be "shut down" by use of the following command:

END.

The END. command must always be the last JAVS command. The actions which occur as a consequence of this command are important to the JAVS user in only one regard--when LIBNEW is completed and is to be saved for future runs. The wrapup sequence provides necessary information about the contents of the new library. The sample output shows the standard wrapup output, after running BASIC and STRUCTURAL. This report contains a printout of the Module Descriptor Table and library information for LIBOLD(1) and LIBNEW(2).

Rule

This must be the last JAVS command.

Sample Output

```
JAVS WRAPUP...

KNOWN MODULE DESCRIPTOR BLOCKS...

     MODULE    TEXT     PARENT          PROC PROBE        EXEC FIRST  WORD                          OS    PARMS DIRECT COMP
NO.  NAME      NAME     NAME      TYPE  SCOPE LINS STMTS  STMTS EXEC  PAIRS  TOKS  SYMS  SLTS  DMTS  OOPS  BLKS  IN OUT CODE TOT OOP
---------------------------------------------------------------------------------------------------------------------------------
  1 EXCOMPL  EXCOMPL  EXCOMPL   CMPL          0    13     3     3     122    53     1     6     0     0     0     0   0  NO    0   0
  2 EXPROGM  EXPROGM  EXPROGM   PROG  EXT     0    33    14    16     270   192     2    26     2     5    23     0   0  NO    0   0
  3 EXMPL2   EXPROGM  EXPROGM   CLSM  GLBL    0     5     4     4      24    16     1     4     0     1     5     0   0  NO    0   0
  4 EXMPL1   EXPROGM  EXPROGM   PROC  INT     0    44    29     9     279   202     3    21     1    20    84     2   0  NO    0   0
  5 EXMPL3   EXPROGM  EXMPL1    CLSP  LOCL    0     4     4     4      21    14     1     3     0     1     4     0   0  NO    0   0
---------------------------------------------------------------------------------------------------------------------------------

LIBRARY INFORMATION--RUN OF 0731

   LIBRARY     TYPE      DATE   TIMES  LAST      TOTAL      LIBRARY    LIBRARY
NO. NAME       ACCESS    CREATED ALTERED ALTERED  WORDS      MODULES    FRAGMENTS
-----------------------------------------------------------------------------
 1  **NOT OPENED**
 2 TEST         R/W       0731    2     0731     17423        5          35
-----------------------------------------------------------------------------
```

Output Description

The Module Descriptor Block (MDB) summary shown in the sample output, reading left to right, gives the following statistics:

1. Module number on the file
2. Module name
3. START-TERM text name
4. Name of parent module (i.e., the module in which this module is contained)

END. (Cont.)

5. Type of module where

   CMPL is a COMPOOL

   PROG is a Program

   PROC is a Procedure

   CLSM is a CLOSE of global scope

   CLSP is a CLOSE of local scope

   TEXT is a START-TERM text which has not been separated into modules by BASIC

6. Scope of an executable module where

   EXT is a program or an external procedure

   INT is an internal procedure (i.e., within a program or external procedure)

   GLBL is a CLOSE within a program or external procedure

   LOCL is a CLOSE within an internal procedure

7. Number of lines of probed text inserted in the module by INSTRUMENT

8. Number of statements in the module

9. Number of executable statements

10. The statement number of the first executable statement

11. Word pairs which measure the amount of library space occupied by the source text

12. Number of tokens in the module

13. Number of symbols defined in module

14. Number of symbols referenced in module

The remainder of the statistics are more applicable to users who wish to follow a testing strategy of analyzing the complexity of their codes and applying testcases to more comprehensively exercise those modules which exhibit a greater tendency to be error prone. These statistics are listed below:

END. (Cont.)

15. Number of other modules invoked by module

16. Number of DD-paths

17. Number of DD-path/statement block entries

18. Number of formal input parameters (-1 until STRUCTURAL)

19. Number of formal output parameters (-1 if the module is a function)

20. YES if DIRECT code is present in module; NO otherwise

21. Statement-based complexity is a summation of the complexity of all statements in the module*

22. DD-path based complexity is a summation of the complexity of all DD-paths in the module. This is used as an indicator of the structural complexity of the module.*

These statistics can be used by the tester to develop a rational approach to program verification of modules which have the greatest tendency toward error (i.e., those which are most complex).

The Library Information report shown in the sample output includes the name for each library, the type of access, the date created, the total size and other useful information.

*These items are provided for future extensions to JAVS and are not presently computed.

END FOR. END FOR.

Description

This iteration command concludes a block of commands which are repeated for each specified module. There are three sequences of commands which select a number of modules and iterates a block of commands (which cannot contain another iteration). The three forms of command iteration are:

(1)
```
FOR MODULE = <name-1>,...,<name-n>.
(any set of commands)
END FOR.
```

(2)
```
FOR JAVSTEXT.
(any set of commands)
END FOR.
```

(3)
```
FOR LIBRARY.
(any set of commands)
END FOR.
```

Rule

Maximum of 150 modules selected in command iteration loop.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
ASSIST,CROSSREF,LIBRARY.
FOR LIBRARY.
PRINT,MODULE.
ASSIST,PICTURE.
END FOR.
END.
```

FOR JAVSTEXT.

Description

The following sequence selects each known module within the "current text" and iterates a block of commands (which cannot contain another iteration) once for each known module of that text:

```
FOR JAVSTEXT.
(any set of commands)
END FOR.
```

Rule

Maximum of 150 modules selected in this iteration command.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
PRINT,MODULE.
END FOR.
END.
```

FOR LIBRARY.

Description

The following sequence selects each known module on the library and iterates a block of commands (which cannot contain another iteration) once for each module:

```
FOR LIBRARY.
(any set of commands)
END FOR.
```

Rule

Maximum of 150 modules selected for this iteration command.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
PRINT,MODULE.
END FOR.
DEPENDENCE,GROUP,LIBRARY.
END.
```

FOR MODULE = <name>...

FOR MODULE = <name>...

Description

The following sequence selects a number of modules, by name, and iterates a block of commands (which cannot contain another iteration) once for each specified module:

```
FOR MODULE = <name-1>,...,<name-n>.
(any set of commands)
END FOR.
```

Rule

A maximum of 25 modules can be specified.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR MODULE = EXPROGM,EXMPL1.
PRINT,MODULE.
PRINT,DDPATHS.
END FOR.
END.
```

INSTRUMENT.

Description

The INSTRUMENT command with all options at their default values produces the small report in the sample output. The actions performed by the INSTRUMENT command is to record probe text statements on LIBNEW.

Rules

(1) See INSTRUMENT constraints in Sec. 3.4.

(2) The INSTRUMENT options are reset to their default values following execution of this command.

Sample Output

```
JOVIAL AUTOMATED VERIFICATION SYSTEM *** INSTRUMENTATION ***

OPTIONS IN EFFECT . . .
                              DD-PATH PROBE =          PROBE
                              MODULE  PROBE =          PROBM
                              TEST    PROBE =          PROBI

INSTRUMENTING ENTRY POINTS, RETURNS AND DD-PATHS OF MODULE <EXPROGM > OF JAVSTEXT <EXPROGM >
```

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT,MODE = FULL.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

INSTRUMENT,MODE = <option>. INSTRUMENT,MODE = <option>.

Description

INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.
(DEFAULT = DDPATHS.)

This command allows user control over the extent of probe insertion. There are two types of instrumentation: structural and performance.

Structural instrumentation may be requested at the module invocation and return level with the command:

INSTRUMENT,MODE = INVOCATION.

This is useful in developing an invocation and return trace for a set of modules.

Structural instrumentation at the DD-path level is produced with the default option of the command:

INSTRUMENT,MODE = DDPATHS.

The instrumentation for module invocation and return level is also included. This is needed to develop a complete execution trace during test execution.

JAVS performance instrumentation is produced with the command:

INSTRUMENT,MODE = DIRECTIVES.

This is used when execution-time monitor output is to be generated as the result of user-supplied JAVS computation directives present in the JOVIAL source text (see Sec. 1.5).

Probe text for both structural instrumentation and performance instrumentation is produced with the command:

INSTRUMENT,MODE = FULL.

MODE = FULL is, in effect, a combination of MODE = DDPATHS and MODE = DIRECTIVES.

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
INSTRUMENT,MODE = INVOCATION.
INSTRUMENT.
END FOR.
END.
```

INSTRUMENT,MODE = <option>. (Cont.)

(2)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
INSTRUMENT,MODE = INVOCATION.
INSTRUMENT.
FOR MODULE = EXMPL1,EXMPL2.
INSTRUMENT,MODE = FULL.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = EXPROGM,EXMPL1,EXMPL2.
END.
```

INSTRUMENT,PROBE,DDPATH = <name>.

INSTRUMENT,PROBE,DDPATH = <name>.

Description

INSTRUMENT,PROBE,DDPATH = <probe-name>. (DEFAULT = PROBE.)

This command is included for use only when it is necessary to change the name of the DD-path data collection routine to avoid conflict. If the default name PROBE is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
INSTRUMENT,PROBE,DDPATH = DDPATH.
INSTRUMENT.
END FOR.
END.
```

INSTRUMENT,PROBE,MODULE = <name>.

INSTRUMENT,PROBE,MODULE = <name>.

Description

INSTRUMENT,PROBE,MODULE = <invocation-name>. (DEFAULT = PROBM.)

This command is included for use only when it is necessary to change the name of the module invocation-and-return data collection routine to avoid conflict. If the default name PROBM is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT,MODE = INVOCATION.
INSTRUMENT,PROBE,MODULE = INVOR.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

INSTRUMENT,PROBE,TEST = <name>.

INSTRUMENT,PROBE,TEST = <name>.

Description

INSTRUMENT,PROBE,TEST = <test-name>. (DEFAULT = PROBI.)

This command is included for use only when it is necessary to change the name of the test-beginning and test-end data collection routine to avoid conflict. If the default name PROBI is used in the JOVIAL source being processed, a different name must be selected (and appropriate modifications made in Test Execution).

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR MODULE = EXPROGM,EXMPL1.
INSTRUMENT,STARTTEST = EXMPL1,EXPROGM,0.
INSTRUMENT,STOPTEST = EXPROGM,EXPROGM,0.
INSTRUMENT,PROBE,TEST = TESTPB.
INSTRUMENT,MODE = FULL.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

INSTRUMENT,STARTTEST = <options>.

INSTRUMENT,STARTTEST = <options>.

Description

In order to identify test cases and control the recording of data on the AUDIT file, the user must supply invocations to the data collection routine, PROBI. The invocations can be manually inserted prior to Test Execution, or they can be automatically inserted during instrumentation (see Sec. 2.6.1).

The INSTRUMENT,STARTTEST command causes JAVS to insert an invocation to PROBI for identifying a new test case. The command is of the form:

INSTRUMENT,STARTTEST = <m-name>,<t-name>,<no.>{,<TESNAM>,<TFLAG>}.

where

m-name = module name

t-name = JAVSTEXT name

no. = statement number

TESNAM = test case identifier DEFAULT = 8H(CASE)

TFLAG = test file control value DEFAULT = 2

The command options must be given in the order shown above. The user must specify the module and JAVSTEXT names in which the PROBI invocation is to be inserted. The user must also specify the statement number (using the statement number presented in a PRINT,MODULE listing) before which the invocation is to be inserted. The user may specify the statement number to be 0. This results in placing the PROBI call immediately prior to the first software probe (at the first executable statement).

TESNAM and TFLAG are the two input parameters to PROBI. If these parameters are not specified in this command, the default values will be used, causing tracing of module invocations and returns. TESNAM may be up to 8 characters; TFLAG may be 1, 2, or 3. A maximum of ten INSTRUMENT,STARTTEST commands can be used for a single INSTRUMENT verb. The module and JAVSTEXT names may be different in each command.

Example Command Sets

See the examples for the INSTRUMENT,STOPTEST command.

INSTRUMENT,STOPTEST = <options>. INSTRUMENT,STOPTEST = <options>.

Description

This command is used to automatically insert calls to data collection routine PROBI to terminate data collection on the AUDIT file. Its utilization is similar to the INSTRUMENT,STARTTEST command. The command's form is:

INSTRUMENT,STOPTEST = <m-name>,<t-name>,<no.>.

where

m-name = module name

t-name = Javstext name

no. = statement number

Only one INSTRUMENT,STOPTEST command may be used for a single INSTRUMENT verb. The PROBI call to terminate testing will be placed immediately prior to the specified statement number or, if the statement number is specified as 0, immediately following the software probe at each exit from the module.

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT,STARTTEST = EXPROGM,EXPROGM,16.
INSTRUMENT,STARTTEST = EXPROGM,EXMPL1,0.
INSTRUMENT,STOPTEST = EXPROGM,EXPROGM,31.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

This command set will instrument DD-paths, invocations and returns for all modules in JAVSTEXT EXPROGM. In addition, the calls to PROBI to initiate a new test case will be automatically inserted immediately prior to statement 16 in module EXPROGM and the first executable statement in module EXMPL1. The test termination PROBI invocation will be automatically inserted prior to statement 31 in module EXPROGM. The default TESNAM and TFLAG will be used; thus the AUDIT file will contain only a module invocation trace but coverage data at the DD-path level. The instrumented source text for JAVSTEXT EXPROGM will be written to file LPUNCH, as well as stored on LIBNEW.

(2)
```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
```

INSTRUMENT,STOPTEST = <options>. (Cont.)

```
INSTRUMENT,MODE = INVOCATIONS.
INSTRUMENT,STARTTEST = EXPROGM,EXPROGM,0,EXPROGM,1.
INSTRUMENT,STOPTEST = EXPROGM,EXPROGM,0.
INSTRUMENT.
FOR MODULE = EXMPL1,EXMPL2,EXMPL3.
INSTRUMENT,STARTTEST = EXMPL1,EXPROGM,9,EXMPL1,3.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

This command set will instrument module EXPROGM for invocations and returns and will instrument the remaining modules in the JAVSTEXT for DD-paths as well. The test case initiations will be automatically inserted in module EXPROGM at its first executable statement and in module EXMPL1 at statement 9. The termination of all test cases will be inserted following the last executable statement in EXPROGM.

The name of each test case will be the name of the module containing the PROBI call. During Test Execution, the AUDIT file will contain no trace for module EXPROGM, but it will have a DD-path and module invocation/return trace for the remaining modules as soon as PROBI is invoked with TFLAG = 3. The AUDIT file will contain module invocation/return summary data for all modules in the JAVSTEXT and DD-path summary (coverage) data for modules EXMPL1, EXMPL2, and EXMPL3.

The instrumented source text for JAVSTEXT EXPROGM will be written to file LPUNCH.

JAVSTEXT = <name>.

Description

Some JAVS commands require specification of the START-TERM text sequence upon which computations are to be performed. START-TERM texts are known to JAVS by their names. The following command makes the "current text" be the named one:

JAVSTEXT = <text-name>.

All subsequent commands (if they refer to a specific text) are applied to the specified text. There can be any number of JAVSTEXT = commands. A new JAVSTEXT = command replaces and supersedes a previous JAVSTEXT = command.

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
END.
```

(2)
```
ALTER LIBRARY = REFMAN.
JAVSTEXT = EXCOMPL.
MODULE = EXCOMPL.
PRINT,MODULE.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
STRUCTURAL.
PRINT,MODULE.
END FOR.
END.
```

Description

The MERGE command can be used (1) to add new modules to an existing library, and (2) to replace old modules with new modules having the same module name and same text name. The command is

```
MERGE.
```

The MERGE command merges a LIBOLD with a LIBNEW. After merging, LIBNEW contains all modules on the original LIBNEW, plus all modules on the LIBOLD that do not have the same module name and same text name as a module on the LIBNEW. Both LIBOLD and LIBNEW must be specified in the library commands (see Sec. 4 and 4.1).

Rule

Maximum of 250 modules in the combined library.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
CREATE LIBRARY = NEWLIB.
START.
BASIC.
MERGE.
FOR LIBRARY.
PRINT,MODULE.
END FOR.
END.
```

This command set will create a new library containing the syntax analysis for the source text on file READER.* After the library is built, the contents of the old library (LIBOLD), excluding any modules with the same module and text names as those on LIBNEW, will be copied onto the new library. This command set can be used for replacing a module on a library with a modified version of that module. In this case LIBNEW will contain the same module names as LIBOLD contains, but LIBNEW will have the modified module(s). Following the library merge, the above command set will print all of the modules on LIBNEW.

(2)
```
OLD LIBRARY = REFMAN.
ALTER LIBRARY = OLIBE.
START.
MERGE.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

*See Table 1.1 on page 1-4.

MERGE. (Cont.)

This command set assumes that syntax analysis has been performed on libraries REFMAN and OLIBE. All of the modules in REFMAN (on LIBOLD), not already contained in OLIBE, will be copied to LIBNEW. Structural analysis will be performed on OLIBE. REFMAN remains unchanged.

(3)
```
OLD LIBRARY = LIBE1.
ALTER LIBRARY = LIBE2.
START.
MERGE.
END.
```

This command set assumes that at least syntax analysis has been performed on both libraries. LIBOLD will be copied onto LIBNEW (LIBE2), excluding any modules in LIBE1 which already exist in LIBE2. No other processing is performed.

MODULE = <name>.

Description

Modules are known to JAVS by their names and the text to which they belong. The following makes the "current module" within the "current JAVSTEXT" be the named one:

```
MODULE = <name>.
```

All subsequent commands (if they refer to a specific module) are applied to the specified module within the "current text." If no current JAVSTEXT has been specified, the module selector will locate the module using only its name.

Rule

If there are duplicate module names, in different JAVSTEXTs, the module selector will choose the last one on the library.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
ASSIST,PICTURE.
MODULE = EXMPL2.
PRINT,MODULE.
END.
```

This command set will produce the control flow picture for module EXMPL1 and will print the source text for EXMPL2.

OLD LIBRARY = <name>.

OLD LIBRARY = <name>.

Description

This command specifies that LIBOLD is to be used during the current run. The name identifying the library should be that same as when the library was created. LIBOLD is a read-only library.

Rule

If the name is different, an informative message is printed, but processing continues.

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
DEPENDENCE,GROUP,LIBRARY.
FOR LIBRARY.
DEPENDENCE,PRINT,INVOKES.
PRINT,MODULE.
END FOR.
END.
```

Description

This command produces a detailed listing of the JAVS information describing each DD-path within the current module.

The properties of each DD-path shown are:

- The DD-path number
- The first statement on the DD-path
- The last statement on the DD-path
- The number of edges (i.e., non-consecutive statement segments) which comprise the DD-path
- The complexity of the DD-path*
- The highest level for the DD-path*
- Parallelism indicator*
- The number of times the DD-path has been tested*
- The number of statements on the DD-path
- DS Table index of the first entry in the list of statements on the DD-path
- The list of statements on the DD-path in execution order.

*The RADC version of JAVS does not compute a value for this entry.

PRINT,DDP. (Cont.)

Sample Output

```
DD-PATH TABLE LISTING

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

      1ST END          PRD  TOP PAR  X  NO. DS
NO.   ST  ST EDG COM  IND  LVL DD- TST ST. INDEX  STATEMENTS ON DD-PATH
-------------------------------------------------------------------------------
  1    1   9   2   0    1    0   0   0   3    1     1   2   9
  2    9  21   9   0    1    0   0   0  10    4     9  10  11  12  13  15  16  18  19  21
  3    9  21   8   0    2    0   0   0   9   14     9  11  12  13  15  16  18  19  21
  4   21  28   3   0    1    0   0   0   4   23    21  22  25  28
  5   21  23   1   0    2    0   0   0   2   27    21  23
  6   23  28   3   0    1    0   0   0   4   29    23  24  25  28
  7   28  30   2   0    1    0   0   0   3   33    28  27  30
  8   28  30   2   0    2    0   0   0   3   36    28  27  30
  9   28  30   2   0    3    0   0   0   3   39    28  27  30
 10   28  30   3   0    4    0   0   0   4   42    28  27  29  30
 11   28  30   3   0    5    0   0   0   4   46    28  27  29  30
 12   30  39   4   0    1    0   0   0   5   50    30  31  36  38  39
 13   30  32   1   0    2    0   0   0   2   55    30  32
 14   32  39   4   0    1    0   0   0   5   57    32  33  36  38  39
 15   32  34   1   0    2    0   0   0   2   62    32  34
 16   34  39   4   0    1    0   0   0   5   64    34  35  36  38  39
 17   39  21   3   0    1    0   0   0   4   69    39  18  19  21
 18   39  41   1   0    2    0   0   0   2   73    39  41
 19   41  21   7   0    1    0   0   0   8   75    41  12  13  15  16  18  19  21
 20   41  44   1   0    2    0   0   0   2   83    41  44
-------------------------------------------------------------------------------
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,MODULE.
PRINT,DDP.
PRINT,DDPATHS.
END FOR.
END.
```

PRINT,DDPATHS.

Description

For the current module, this command produces a detailed listing of the source text for the statements which begin a DD-path with the DD-path description included. The report is similar in format to the report from the PRINT, MODULE command. Only those statements which begin a DD-path or are crucial to the interpretation of a DD-path (e.g., FOR, SWITCH) are displayed.

Sample Output

```
MODULE DD-PATH DEFINITION LISTING

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

 NO.  LVL          STATEMENT                                   DD-PATHS GENERATED
-------------------------------------------------------------------------------------
   1 ( 0)          PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $
                                                          ** DD-PATH   1 IS PROCEDURE ENTRY
                    . . . .
   9 ( 1)           IF LIMIT1 GQ 100 $
                                                          ** DD-PATH   2 IS TRUE BRANCH
                                                          ** DD-PATH   3 IS FALSE BRANCH
                    . . . .
  12 ( 1)           FOR I = 1 , 1 , LIMIT1 $
  13 ( 2)             BEGIN
                      . . . .
  18 ( 2)             FOR J = 1 , 1 , LIMIT2 $
  19 ( 3)               BEGIN
                        . . . .
  21 ( 3)               IFEITH J LQ 3 $
                                                          ** DD-PATH   4 IS TRUE BRANCH
                                                          ** DD-PATH   5 IS FALSE BRANCH
                        . . . .
  23 ( 3)               ORIF 1 $
                                                          ** DD-PATH   6 IS TRUE BRANCH
                        . . . .
  25 ( 3)               END
                        . . . .
  27 ( 3)               SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
  28 ( 3)               GOTO PICK ($ INDXS - 1 $) $
                                                          ** DD-PATH   7 IS SWITCH OUTWAY   1
                                                          ** DD-PATH   8 IS SWITCH OUTWAY   2
                                                          ** DD-PATH   9 IS SWITCH OUTWAY   3
                                                          ** DD-PATH  10 IS SWITCH OUTWAY   4
                                                          ** DD-PATH  11 IS SWITCH OUTWAY   5
                        . . . .
  30 ( 3)       LABEL1.
                        IFEITH RESULT LS 4 $
                                                          ** DD-PATH  12 IS TRUE BRANCH
                                                          ** DD-PATH  13 IS FALSE BRANCH
                        . . . .
  32 ( 3)               ORIF RESULT EQ 4 $
                                                          ** DD-PATH  14 IS TRUE BRANCH
                                                          ** DD-PATH  15 IS FALSE BRANCH
                        . . . .
  34 ( 3)               ORIF 1 $
                                                          ** DD-PATH  16 IS TRUE BRANCH
                        . . . .
  36 ( 3)               END
                        . . . .
  39 ( 3)               END
                                                          ** DD-PATH  17 IS LOOP ON FOR AGAIN
                                                          ** DD-PATH  18 IS ESCAPE FOR LOOP
                      . . . .
  41 ( 2)             END
                                                          ** DD-PATH  19 IS LOOP ON FOR AGAIN
                                                          ** DD-PATH  20 IS ESCAPE FOR LOOP
-------------------------------------------------------------------------------------
```

PRINT,DDPATHS. (Cont.)

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
PRINT,DDPATHS.
END FOR.
END.
```

PRINT,DMT.

Description

This command produces a listing of the internal dependent module table (DMT) descriptions. The table contains an entry for each invocation contained in the current module. Pointers to other internal tables (e.g., the SB and SLT) are included. This output is intended for JAVS system maintenance only.

Sample Output

```
DEPENDENT MODULE TABLE LISTING

MODULE <EXPROGM >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

NO.   EXTERNAL INVOKED   SLT NUMBER   STMT OF INVOCATION   WRD-PR OF INVOCATION   SERIAL INVOCATION
-----------------------------------------------------------------------------------------------
  1   EXMPL1                    24                   24                     1                   1
  2   EXMPL2                    26                   27                     3                   1
-----------------------------------------------------------------------------------------------
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
PRINT,DMT.
END.
```

PRINT,JAVSTEXT = <name>.

PRINT,JAVSTEXT = <name>.

Description

This command produces a listing of the source statements in the named START-TERM text as they were originally stored on the library (see Sec. 1.4). The sample output is after STRUCTURAL processing. The salient features are:

- The statement number assigned by JAVS
- The level of nesting of the statement
- The label(s) and text of the statement indented to show nesting level (e.g., FOR, IF, and IFEITH hierarchy)
- The numbers of the DD-paths which are identified and assigned by JAVS
- Statement type for most control statements

PRINT,JAVSTEXT = <name>. (Cont.)

Sample Output (Excerpt)

```
JAVSTEXT <EXPROGM > LISTING

NO.  LVL            STATEMENT                                                    DD-PATHS        CONTROL
---------------------------------------------------------------------------------------------------------
  1 ( 0)            **, JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) **
  2 ( 0)            START
  3 ( 0)            $
  4 ( 0)            ** JOVIAL SIMPLE TEST PROGRAM **
  5 ( 0)            ITEM ID H 4 $
  6 ( 0)            ITEM ITER1 I 24 S $
  7 ( 0)            ITEM ITER2 I 24 S $
  8 ( 0)            ITEM ITER1A H 4 $
  9 ( 0)            ITEM ITER2A H 4 $
 10 ( 0)            OVERLAY ITER1 = ITER1A $
 11 ( 0)            OVERLAY ITER2 = ITER2A $
 12 ( 0)            ITEM CARD H 80 $
 13 ( 0)            FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
 14 ( 0)            FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
 15 ( 0)            MONITOR ID , ITER1A , ITER2A $
 16 ( 0)            MESSAG = MSG1 $                                              ( 1)
 17 ( 0)            OUTPUT PRINTR MESSAG $                                                       I/O
 18 ( 0)         BG.                                                                             I/O <------
                    INPUT READER CARD $
 19 ( 0)            IF READER NQ V(EOF) $                                        ( 2- 3)         IF
 20 ( 1)              BEGIN
 21 ( 1)              BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
 22 ( 1)              BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
 23 ( 1)              BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
 24 ( 1)              EXMPL1 ( ITER1 , ITER2 ) $                                                   INV
  1 ( 1)              CLOSE EXMPL2 $                                               ( 1)
  2 ( 1)            ** MAIN CLOSE **
  3 ( 2)                BEGIN
  4 ( 2)                ITER1 = 1 $
  5 ( 2)                ITER2 = 1 $
  6 ( 2)                END
 25 ( 1)            ** EXMPL2 **
 26 ( 1)              IF ITER1 GQ 100 $                                          ( 4- 5)         IF
 27 ( 2)                GOTO EXMPL2 $                                                            INV
 28 ( 1)              GOTO BG $                                                                       ------>
 29 ( 1)              END
 30 ( 0)            ** IF **
 31 ( 0)            STOP $
  1 ( 0)            PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $                            ( 1)
  2 ( 1)              BEGIN
  3 ( 1)              ITEM LIMIT1 I 24 S $
  4 ( 1)              ITEM LIMIT2 I 24 S $
  5 ( 1)              ARRAY FILL 100 I 24 S $
  6 ( 1)              ITEM RESULT I 24 S $
  7 ( 1)              ITEM INDXS I 24 S $
  8 ( 1)            **, TRACE , RESULT **
  9 ( 1)              IF LIMIT1 GQ 100 $                                         ( 2- 3)         IF
 10 ( 2)                LIMIT1 = 99 $
 11 ( 1)              RESULT = 4 $
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
PRINT,JAVSTEXT = EXPROGM.
END.
```

PRINT,JAVSTEXT = <name>,
INSTRUMENTED = <option>.

PRINT,JAVSTEXT = <name>,
INSTRUMENTED = <option>.

Description

This command has two forms:

PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = ALL. or

PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>,<name-2>,...,
<name-n>.

The command produces a listing of the instrumented source (i.e., results of INSTRUMENT)* in the named START-TERM text. The first form is used when all modules of the text are listed with instrumentation inserted. The second form is used when specified modules in the text are to have instrumentation inserted and all other modules in the text are not instrumented. Any module which has not been processed by INSTRUMENT will be shown in its original form. The modules must be listed in the same order in which they appear on the library. The salient features of the output are:

- The statement number within the module of the original source statement assigned by JAVS
- The level of nesting of the statement or an indicator (P) if the statement shown is a probe (i.e., generated by INSTRUMENT)
- The label(s) and text of the statement indented to show nesting level.

The code generated by the instrumentation does not affect the logic of the user's program. In some instances (as with the last statement 11 in the sample output), a P is placed by the user's own code. This happens when the instrumentation code modification, stored in the library, is identical to the user's statement.

Rule

Specified modules must be in the same order in which they appear on the library.

*The instrumented text consists of the source text interspersed with the probe text generated by INSTRUMENT.

PRINT,JAVSTEXT = <name>,INSTRUMENTED = <option>. (Cont.)

Sample Output


```
JAVSTEXT <EXPROGM > LISTING

NO.  LVL              STATEMENT                                                    DD-PATHS   CONTROL
---------------------------------------------------------------------------------------------------
  1 ( 0)              **, JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) **
  2 ( 0)              START
  3 ( 0)              $
  4 ( 0)              ** JOVIAL SIMPLE TEST PROGRAM **
  5 ( 0)              ITEM ID H 4 $
  6 ( 0)              ITEM ITER1 I 24 S $
  7 ( 0)              ITEM ITER2 I 24 S $
  8 ( 0)              ITEM ITER1A H 4 $
  9 ( 0)              ITEM ITER2A H 4 $
 10 ( 0)              OVERLAY ITER1 = ITER1A $
 11 ( 0)              OVERLAY ITER2 = ITER2A $
 12 ( 0)              ITEM CARD H 80 $
 13 ( 0)              FILE READER H 0 F 80 V(OK) V(EOF) TAPE5 $
 14 ( 0)              FILE PRINTR H 0 F 128 V(OK) V(EOF) TAPE6 $
 15 ( 0)              MONITOR ID , ITER1A , ITER2A $
 16 P              PROBM    ( BM(EXPROGM ) , BM(EXPROGM ) ,    5 ) $
 16 P              PROBE    (BM(EXPROGM ), BM(EXPROGM ),    11 $
 16 P              MESSAG =   MSG1    $
 17 ( 0)               OUTPUT PRINTR MESSAG $                                                          I/O
 18 ( 0)            BG.                                                                               I/O <------
                       INPUT READER CARD $
 19 P              IFEITH    READER NQ V(EOF) $
 19 P                BEGIN
 19 P                PROBE    (BM(EXPROGM ), BM(EXPROGM ),    2) $
 20 ( 2)                   BEGIN
 21 ( 2)                   BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
 22 ( 2)                   BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
 23 ( 2)                   BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
 24 ( 2)                   EXMPL1 ( ITER1 , ITER2 ) $                                                  INV
  1 ( 2)                   CLOSE EXMPL2 $                                                 (   1)
  2 ( 2)                ** MAIN CLOSE **
  3 ( 3)                      BEGIN
  4 P                    PROBM    ( BM(EXMPL2  ) , BM(EXPROGM ) ,    1 ) $
  4 P                    PROBE    (BM(EXMPL2  ), BM(EXPROGM ),    1) $
  4 P                    ITER1   =   1   $
  5 ( 3)                       ITER2 = 1 $
  6 P                    PROBM    ( BM(EXMPL2  ) , BM(EXPROGM ) ,    0 ) $
  6 P                    END
 25 ( 2)                ** EXMPL2 **
 26 P                  IFEITH    ITER1 GQ 100 $
 26 P                    BEGIN
 26 P                    PROBE    (BM(EXPROGM ), BM(EXPROGM ),    4) $
 27 ( 3)                      GOTO EXMPL2 $                                                           INV
 28 P                    END
```

Sample Command Set

```
ALTER LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

PRINT,MODULE.

Description

This command produces a detailed listing of the source text of the current module. The sample output is after STRUCTURAL processing. The salient features are:

- The statement number assigned by JAVS
- The level of nesting of the statement
- The label(s) and text of the statement indented to show nesting level (e.g., FOR, IF, and IFEITH hierarchy)
- The numbers of the DD-paths which are identified and assigned by JAVS
- Statement type for most control statements

PRINT,MODULE. (Cont.)

Sample Output

```
MODULE STATEMENT LISTING

MODULE <EXMPL1  >. JAVSTEXT <EXPROGM >. PARENT MODULE <EXPROGM >

 NO.  LVL            STATEMENT                                          DD-PATHS        CONTROL
---------------------------------------------------------------------------------------------------
   1 ( 0)          PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $                    ( 1)
   2 ( 1)            BEGIN
   3 ( 1)            ITEM LIMIT1 I 24 S $
   4 ( 1)            ITEM LIMIT2 I 24 S $
   5 ( 1)            ARRAY FILL 100 I 24 S $
   6 ( 1)            ITEM RESULT I 24 S $
   7 ( 1)            ITEM INDXS I 24 S $
   8 ( 1)          ##. TRACE , RESULT ##
   9 ( 1)            IF LIMIT1 GQ 100 $                                 ( 2- 3)  IF
  10 ( 2)              LIMIT1 = 99 $
  11 ( 1)            RESULT = 4 $
  12 ( 1)            FOR I = 1 , 1 , LIMIT1 $                                    FOR3
  13 ( 2)              BEGIN
  14 ( 2)          ##. EXPECT , RESULT = 1 , 5 ##
  15 ( 2)              FILL ($ I - 1 $) = I $
  16 ( 2)              RESULT = ( RESULT + I ) / I $
  17 ( 2)          ##. ASSERT , RESULT GR 10 ##
  18 ( 2)              FOR J = 1 , 1 , LIMIT2 $                                FOR3
  19 ( 3)                BEGIN
  20 ( 3)          ## CLOSE ##
  21 ( 3)                IFEITH J LQ 3 $                                 ( 4- 5)  IFE1
  22 ( 4)                  INDXS = J $
  23 ( 3)                ORIF 1 $                                        ( 6)     ORIF
  24 ( 4)                  INDXS = 4 $
  25 ( 3)                END
  26 ( 3)           ## IFEITH ##
  27 ( 3)                SWITCH PICK = ( LABEL1 , LABEL1 , LABEL1 , LABEL2 ) $
  28 ( 3)                GOTO PICK ($ INDXS - 1 $) $                     ( 7- 11)      ------>
  29 ( 3)          LABEL2.                                                          INV <------
                         GOTO EXMPL3 $
  30 ( 3)          LABEL1.                                               ( 12- 13)  IFE1<------
                         IFEITH RESULT LS 4 $
  31 ( 4)                  MESSAG = MSG2 $
  32 ( 3)                ORIF RESULT EQ 4 $                              ( 14- 15)  ORIF
  33 ( 4)                  MESSAG = MSG3 $
  34 ( 3)                ORIF 1 $                                        ( 16)      ORIF
  35 ( 4)                  MESSAG = MSG4 $
  36 ( 3)                END
  37 ( 3)           ## IFEITH ##                                                    I/O
  38 ( 3)                OUTPUT PRINTR MESSAG $
  39 ( 3)                END                                             ( 17- 18)
  40 ( 2)           ## J ##                                              ( 19- 20)
  41 ( 2)             END
  42 ( 1)           ## I ##
  43 ( 1)           ##. OFFTRACE , RESULT ##
  44 ( 1)             END
---------------------------------------------------------------------------------------------------
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,MODULE.
END FOR.
END.
```

PRINT,SB.

Description

This command produces a detailed listing of how the statements within the current module are stored in the statement block (SB) on the library. Each statement is shown in token and word-pair form. The upper line contains the text of the statement and the lower line indicates the token classification. This output is intended for JAVS system maintenance only.

Sample Output

```
STATEMENT BLOCK LISTING

MODULE <EXPROGM >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

STATEMENT    1

##,   JAVS  TEXT        EXPR  OGM   COMP  UTE   (     EXCO  MPL   )     ##
DELI  IDEN  CONT  ENDT  IDEN  ENDT  IDEN  ENDT  DELI  IDEN  ENDT  DELI  DELI

STATEMENT    2

STAR  T
KEYW  ENDT

STATEMENT    3

$
DELI

STATEMENT    4

##    JOVI  AL    SIMP  LE    TEST        PROG  RAM   ##
COMT  COMT  ENDT  COMT  ENDT  COMT  ENDT  COMT  ENDT  COMT

STATEMENT    5

ITEM        ID    H     4     $
KEYW  ENDT     1  ABBR        DELI

STATEMENT    6

ITEM        ITER  1     I     24    S     $
KEYW  ENDT     2  ENDT  ABBR        ABBR  DELI

STATEMENT    7

ITEM        ITER  2     I     24    S     $
KEYW  ENDT     3  ENDT  ABBR        ABBR  DELI
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXPROGM.
PRINT,SB.
END.
```

PRINT,SDB.

Description

This command produces a detailed listing of the internal statement descriptors generated and saved by JAVS. This output is intended for researchers or sophisticated JAVS users only.

Sample Output

STATEMENT DESCRIPTOR LISTING

MODULE <EXMPL1 >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

NO.	STMT CLASS	TYPE CODE	SEQ NUM	WD-PRS	SB INDEX	NO. LABELS	LABEL WD-PRS	INTER-MTH PTR	INTRA-MTH PTR	COMPLX
1	PROC	100	38	12	1	0	0	0	5	0
2	BEGN	100	39	2	25	0	0	0	0	0
3	ITEM	1	40	8	29	0	0	0	0	0
4	ITEM	1	41	8	45	0	0	0	0	0
5	ARRY	1	42	9	61	0	0	0	0	0
6	ITEM	1	43	8	79	0	0	0	0	0
7	ITEM	1	44	8	95	0	0	0	0	0
8	DIRT	1	45	7	111	0	0	0	0	0
9	IF	200	46	6	125	0	0	11	0	0
10	ASMT	100	47	5	137	0	0	0	3	0
11	ASMT	100	48	5	147	0	0	0	3	0
12	FOR3	300	49	10	157	0	0	41	0	0
13	BEGN	100	50	2	177	0	0	0	0	0
14	DIRT	1	51	11	181	0	0	0	0	0
15	ASMT	100	52	10	203	0	0	0	8	0
16	ASMT	100	53	12	223	0	0	0	3	0
17	DIRT	1	54	9	247	0	0	0	0	0
18	FOR3	300	55	10	265	0	0	39	0	0
19	BEGN	100	56	2	285	0	0	0	0	0
20	COMT	1	62	4	289	0	0	0	0	0
21	IFE1	200	63	6	297	0	0	23	0	0
22	ASMT	100	64	5	309	0	0	0	3	0
23	ORIF	500	65	4	319	0	0	25	0	0
24	ASMT	100	66	5	327	0	0	0	3	0
25	END	100	67	1	337	0	0	21	0	0
26	COMT	1	68	4	339	0	0	0	0	0
27	INSW	1	69	19	347	0	0	0	4	0
28	GTSW	100	70	11	385	0	0	27	0	0
29	GTCL	100	71	8	407	1	3	0	0	0
30	IFE1	200	72	10	423	1	3	32	0	0
31	ASMT	100	73	6	443	0	0	0	3	0
32	ORIF	200	74	7	455	0	0	34	0	0
33	ASMT	100	75	6	469	0	0	0	3	0
34	ORIF	500	76	4	481	0	0	36	0	0
35	ASMT	100	77	6	489	0	0	0	3	0
36	END	100	78	1	501	0	0	30	0	0
37	COMT	1	79	4	503	0	0	0	0	0
38	OUTP	100	80	7	511	0	0	0	0	0
39	END	400	81	1	525	0	0	18	0	0
40	COMT	1	82	3	527	0	0	0	0	0
41	END	400	83	1	533	0	0	12	0	0
42	COMT	1	84	3	535	0	0	0	0	0
43	DIRT	1	85	8	541	0	0	0	0	0
44	END	100	86	1	557	0	0	0	0	0

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
MODULE = EXMPL1.
PRINT,SDB.
END.
```

Description

This command produces a listing of the symbol locator table (SLT) (i.e., a list of the symbols referenced within the current module). This list includes those identifiers and constants which appear in the source text which are essential for JAVS processing. The first eight characters of the symbol are stored and pointers to other JAVS-internal tables are included. This output is intended for JAVS system maintenance only.

Sample Output

SYMBOL LOCATION TABLE LISTING

MODULE <EXMPL1 >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

NO.	SYMBOL	MODULE	JAVSTEXT	STB	FIRST	LAST	NUMBER		
1	EXMPL1	EXPROGM	EXPROGM	2	1	1	1	2	82
2	LIMIT1	EXMPL1	EXPROGM	-1	1	12	5	4	93
3	LIMIT2	EXMPL1	EXPROGM	-1	1	18	3	4	96
4	FILL	EXMPL1	EXPROGM	-5	5	15	2	4	99
5	100	EXMPL1	EXPROGM	-5	5	5	1	4	103
6	RESULT	EXMPL1	EXPROGM	-6	6	32	6	4	106
7	INDXS	EXMPL1	EXPROGM	-7	7	28	4	4	109
8	100	EXPROGM	EXPROGM	-26	9	9	1	2	90
9	99	EXMPL1	EXPROGM	-10	10	10	1	4	112
10	4	EXPROGM	EXPROGM	-21	11	32	4	2	73
11	1	EXMPL2	EXPROGM	-4	12	34	8	3	87
12	3	EXMPL1	EXPROGM	-21	21	21	1	4	120
13	PICK	EXMPL1	EXPROGM	1	27	28	2	4	123
14	LABEL1	EXMPL1	EXPROGM	2	27	30	4	4	125
15	LABEL2	EXMPL1	EXPROGM	3	27	29	2	4	127
16	EXMPL3	EXMPL3	EXPROGM	1	29	29	1	5	115
17	MESSAG	EXCOMPL	EXCOMPL	-11	31	38	4	1	15
18	MSG2	EXCOMPL	EXCOMPL	-8	31	31	1	1	6
19	MSG3	EXCOMPL	EXCOMPL	-9	33	33	1	1	9
20	MSG4	EXCOMPL	EXCOMPL	-10	35	35	1	1	12
21	PRINTR	EXPROGM	EXPROGM	-14	38	38	1	2	55

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,SLT.
END FOR.
END.
```

Description

This command produces a listing of the detailed, internal symbol table (STB) descriptors for each symbol defined in the current module. Normally the list includes only those symbols essential to structural analysis of the module. A BASIC processing option causes all symbols defined in the module to be entered in the STB (see BASIC SYMBOLS = ON). The properties of each symbol and pointers to other JAVS-internal tables are included. This output is intended for JAVS system maintenance only.

Sample Output

```
SYMBOL TABLE LISTING

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

NO.  SYMBOL    1   2    3    4   5  6  7   8  9  10  11  12 13  14  15 16  17  18 19  20  22    23    24  27  29  30
-----------------------------------------------------------------------------------------------------------------------
  1 PICK       2 LOCL INSW          4  0                      0  0   0   0  0   0   0  0   0  27  EXMPL1            13
  2 LABEL1     1 LOCL LABL          4 14                      0  0   0   0  0   0   0  0   0  30  EXMPL1            14
  3 LABEL2     1 LOCL LABL          4 15                      0  0   0   0  0   0   0  0   0  29  EXMPL1            15
-----------------------------------------------------------------------------------------------------------------------
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,STB.
END FOR.
END.
```

Description

The PROBE macro command has three forms:

(1) PROBE.

(2) PROBE,JAVSTEXT = <text name>.

(3) PROBE,JAVSTEXT = <text name>, MODULE = <name-1>,<name-2>,...,
<name-n>.

Each form of the PROBE macro command generates a series of JAVS standard commands which cause instrumentation to be performed.

The expansion of each form of the PROBE macro into JAVS standard commands is as follows:

(1) JAVSTEXT = *NOJAVS*.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = *NOJAVS*,INSTRUMENTED = ALL.

This form (PROBE.) can be used for a library containing only one start-term sequence, if the JAVSTEXT identification (see Sec. 1.4) was not used. Instrumentation will be performed using the INSTRUMENT default options, and the instrumented source text will be written onto file LPUNCH.

(2) JAVSTEXT = <text-name>.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = ALL.

This form (PROBE,JAVSTEXT = <text name>.) should be used for instrumentation of a specified JAVSTEXT.

(3) JAVSTEXT = <text name>.
FOR MODULE = <name-1>,...,<name-n>.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = <name-1>,...,<name-n>.

This form (PROBE,JAVSTEXT = <text name>, MODULE = <name-1>....) should be used if only selected modules of a given JAVSTEXT are to be instrumented.

---

*Can be used only with the overlay version of JAVS.

PROBE. (Cont.)

With all forms of the PROBE macro, the user can specify automatic insertion of the PROBI (test case initiation and test termination data collection routine) calls by preceding the PROBE command with:

PROBI,STARTTEST = <modname>,<textname>,<stmt. no.>,{,TESNAM}{,TFLAG}.

PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.

These commands are described in Sec. 5 under their own heading. They perform the same function as the INSTRUMENT,STARTTEST and INSTRUMENT,STOPTEST commands. The existence of the PROBI commands aids the macro command processor which will insert the STARTTEST and STOPTEST commands in the generated series of commands.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY,DOCUMENT, PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) See INSTRUMENT constraints in Sec. 3.4.

(2) Maximum of 150 modules selected in the first two forms of the PROBE macro command.

(3) Maximum of 23 modules selected in the third form of the PROBE macro command.

(4) Use PROBI (not INSTRUMENT,STARTTEST and STOPTEST) commands with the PROBE macro.

Example Command Sets

(1)
```
BUILD LIBRARY.
PROBE,JAVSTEXT = EXPROGM.
DOCUMENT.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
```

This command set will create a new library, perform syntax and structural analyses, instrument the JAVSTEXT EXPROGM, document an entire library, and

print the instrumented JAVSTEXT so that the user can see where the PROBI calls should be manually placed.

(2)
```
OLD LIBRARY = TEST.
START.
PROBI,STARTTEST = EXPROGM,EXPROGM,0.
PROBI,STARTTEST = EXMPL1,EXPROGM,9.
PROBI,STOPTEST = EXPROGM,EXPROGM,0.
PROBE,JAVSTEXT = EXPROGM.
```

This command set will instrument JAVSTEXT EXPROGM in the existing library TEST. In addition, the PROBI calls will be automatically inserted. The first PROBI,STARTTEST command will cause an invocation to PROBI to be inserted prior to the first executable statement in module EXPROGM. The second PROBI, STARTTEST command will cause a PROBI invocation to be placed immediately before statement 9 of module EXMPL1. These two PROBI invocations will initiate a new test case whenever the PROBI call is executed.

The PROBI,STOPTEST command will cause the test termination PROBI invocation to be placed at all exits from module EXPROGM. The PROBE macro command will instrument the JAVSTEXT EXPROGM and write the instrumented text, including the invocations to PROBI, onto file LPUNCH.

Note that the library definition and startup commands are included in this example. Even though the default library name is specified, these commands are required because PROBI is not a macro command.

PROBI,STARTTEST = <options>.* PROBI,STARTTEST = <options>.*

Description

This command performs the same function as:

INSTRUMENT,STARTTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Maximum of 10 PROBI,STARTTEST commands with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) Modules specified in PROBI,STARTTEST commands must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

*Can be used only with the overlay version of JAVS.

PROBI,STOPTEST = <options>.* PROBI,STOPTEST = <options>.*

Description

This command performs the same function as:

INSTRUMENT,STOPTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Only one PROBI,STOPTEST command with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) The module specified in the PROBI,STOPTEST command must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

*Can be used only with the overlay version of JAVS.

PUNCH,JAVSTEXT = <name>.

PUNCH,JAVSTEXT = <name>.

Description

This command produces on LPUNCH in card image format the source statements for the named START-TERM text as they were originally stored on the library. The file can then be punched. The resulting cards are in normal JOVIAL format ready for compilation.

A listing of the cards resulting from this command for text EXPROGM is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

Rule

File LPUNCH must be specified in the job control stream.

Sample Output Excerpt

```
      **. JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) **
      START
      $
      ** JOVIAL SIMPLE TEST PROGRAM **
      ITEM ID H 4 $
      ITEM ITER1 I 24 S $
      ITEM ITER2 I 24 S $
      ITEM ITER1A H 4 $
      ITEM ITER2A H 4 $
      OVERLAY ITER1 = ITER1A $
      OVERLAY ITER2 = ITER2A $
      ITEM CARD H 80 $
      FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
      FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
      MONITOR ID , ITER1A , ITER2A $
      MESSAG = MSG1 $
      OUTPUT PRINTR MESSAG $
BG.
      INPUT READER CARD $
      IF READER NQ V(EOF) $
        BEGIN
        BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
        BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
        BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
        EXMPL1 ( ITER1 , ITER2 ) $
        CLOSE EXMPL2 $
      ** MAIN CLOSE **
          BEGIN
          ITER1 = 1 $
          ITER2 = 1 $
          END
      ** EXMPL2 **
        IF ITER1 GQ 100 $
          GOTO EXMPL2 $
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
PUNCH,JAVSTEXT = EXPROGM.
END.
```

PUNCH,JAVSTEXT = <name>,
INSTRUMENTED = <option>.

PUNCH,JAVSTEXT = <name>,
INSTRUMENTED = <option>.

Description

This command has two forms:

PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = ALL. or

PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>,<name-2>,...,
<name-n>.

The command produces on LPUNCH in card image format the source statements for the named START-TERM text as instrumented by INSTRUMENT.* The file can then be punched. The resulting cards are in normal JOVIAL format ready for compilation.

The first form is used when all modules of the text are punched with instrumentation inserted. The second form is used when the specified modules in the text are to have instrumentation inserted and all other modules in the text are not instrumented. Any module which has not been processed by INSTRUMENT will be punched in uninstrumented, reformatted form. The modules must be listed in the same order in which they appear on the library.

A listing of the cards resulting from this command for text EXPROGM is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

Rules

(1) Specified modules must be listed in the same order in which they appear in the library.

(2) File LPUNCH must be specified in the job control stream.

*The instrumented text consists of the source text interspersed with the probe text generated by INSTRUMENT.

PUNCH,JAVSTEXT = <name>,INSTRUMENTED = <option>. (Cont.)

Sample Output (Excerpt)

```
      ##. JAVSTEAT EXPROGM COMPILE ( EXCOMPL ) ##
      START
      $
      ## JOVIAL SIMPLE TEST PROGRAM ##
      ITEM ID H 4 $
      ITEM ITER1 I 24 S $
      ITEM ITER2 I 24 S $
      ITEM ITER1A H 4 $
      ITEM ITER2A H 4 $
      OVERLAY ITER1 = ITER1A $
      OVERLAY ITER2 = ITER2A $
      ITEM CARD H 80 $
      FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
      FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
      MONITOR ID , ITER1A , ITER2A $
      PROGM     ( BM(EXPROGM ) , BM(EXPROGM ) ,     5 ) $
      PROBE     (BM(EXPROGM ), BM(EXPROGM ),     1) $
      MESSAG =    'MSG1'     $
      OUTPUT PRINTR MESSAG $
HG.
      INPUT READER CARD $
      IFEITH     READER NQ V(EOF) $
        BEGIN
        PROBE     (BM(EXPROGM ), BM(EXPROGM ),     2) $
          BEGIN
          BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
          BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
          BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
          EXMPL1 ( ITER1 , ITER2 ) $
          CLOSE EXMPL2 $
      ## MAIN CLOSE ##
            BEGIN
            PROGM     ( BM(EXMPL2  ) , BM(EXPROGM ) ,     1 ) $
            PROBE     (BM(EXMPL2  ), BM(EXPROGM ),     1) $
            ITER1    =    1    $
            ITER2 = 1 $
            PROGM     ( BM(EXMPL2  ) , BM(EXPROGM ) ,     0 ) $
            END
      ## EXMPL2 ##
          IFEITH     ITER1 GQ 100 $
            BEGIN
            PROBE     (BM(EXPROGM ), BM(EXPROGM ),     4) $
            GOTO EXMPL2 $
            END
          ORIF 1 $
            PROBE     (BM(EXPROGM ), BM(EXPROGM ),     5) $
          END
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
END.
```

PUNCH,MODULE.

Description

This command causes the module to be written on LPUNCH in card image format. The file can then be punched. The resulting cards are in normal JOVIAL format ready for insertion in a text for compilation. If the module is the only one in the START-TERM text, no additional manipulation of the punched text is necessary, since the START and TERM statements are included.

A listing of the cards resulting from this command for the module EXMPL1 is shown in the sample output. The statements are reformatted to a uniform structure with indentation.

Rule

File LPUNCH must be specified in the job control stream.

Sample Output

```
      **. JAVSTEXT EXPROGM COMPUTE ( EXCOMPL ) **
      START
      $
      ** JOVIAL SIMPLE TEST PROGRAM **
      ITEM ID H 4 $
      ITEM ITER1 I 24 S $
      ITEM ITER2 I 24 S $
      ITEM ITER1A H 4 $
      ITEM ITER2A H 4 $
      OVERLAY ITER1 = ITER1A $
      OVERLAY ITER2 = ITER2A $
      ITEM CARD H 80 $
      FILE READER H 0 R 84 V(OK) V(EOF) TAPE5 $
      FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6 $
      MONITOR ID , ITER1A , ITER2A $
      MESSAG = MSG1 $
      OUTPUT PRINTR MESSAG $
BG.
      INPUT READER CARD $
      IF READER NQ V(EOF) $
        BEGIN
        BYTE ($ 0 , 4 $) ( ID ) = BYTE ($ 0 , 4 $) ( CARD ) $
        BYTE ($ 0 , 4 $) ( ITER1A ) = BYTE ($ 9 , 4 $) ( CARD ) $
        BYTE ($ 0 , 4 $) ( ITER2A ) = BYTE ($ 19 , 4 $) ( CARD ) $
        EXMPL1 ( ITER1 , ITER2 ) $
      ** EXMPL2 **
        IF ITER1 GQ 100 $
          GOTO EXMPL2 $
        GOTO BG $
        END
      ** IF **
      STOP $
      ** EXMPL1 **
      TERM $
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PUNCH,MODULE.
END FOR.
END.
```

STRUCTURAL.

Description

The STRUCTURAL command with the print option at its default value produces the report output shown in the sample output. The actions performed by the STRUCTURAL command are:

1. To build tables describing the graphical characteristics of the specified module and add them to a library

2. To produce the report shown in the sample output

Rule

See STRUCTURAL constraints in Sec. 3.3.

Sample Output

```
JOVIAL AUTOMATED VERIFICATION SYSTEM    *** SECONDARY MODULE ANALYSIS ***

MODULE  EXMPL1  > OF JAVSTEXT <EXPROGM >.
     MODULE DEPENDENCE TABLE CONSTRUCTED.
     STATEMENT DESCRIPTOR BLOCKS UPDATED.
     DD-PATH TABLE CONTAINS   20 ENTRIES.
```

Example Command Set

```
ALTER LIBRARY = REFMAN.
START.
JAVSTEXT = EXPROGM.
FOR JAVSTEXT.
STRUCTURAL.
END FOR.
END.
```

STRUCTURAL,PRINT = <option>. STRUCTURAL,PRINT = <option>.

Description

STRUCTURAL,PRINT = SUMMARY/DEBUG. (DEFAULT = SUMMARY.)

This command allows user control over the amount of information printed in the report produced during STRUCTURAL processing. Debug output which is useful to JAVS maintenance activities is generated with the command:

STRUCTURAL,PRINT = DEBUG.

Example Command Set

```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
STRUCTURAL,PRINT = DEBUG.
STRUCTURAL.
END FOR.
END.
```

Description

The TEST macro command has two forms:

(1) TEST.

(2) TEST,MODULE = <name-1>,<name-2>,...,<name-n>.

The TEST macro command generates a series of JAVS ANALYZER commands for post-test analysis.

The expansion of the two forms of TEST is as follows:

(1)
```
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,MODTRACE.
ANALYZER.
```

(2)
```
ANALYZER,MODULE = <name-1>,...,<name-n>.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,MODTRACE.
ANALYZER.
```

Additional post-test analysis reports can be produced by preceding the TEST macro with ANALYZER process option commands. In this case, library identification and startup commands must be included in the command set.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY,DOCUMENT, PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) The ANALYZER,DDPTRACE standard command cannot be used in conjunction with the TEST macro.

---

*Can be used only with the overlay version of JAVS.

TEST. (Cont.)

(2) ANALYZER option commands must precede the TEST macro, if they are used.

(3) See ANALYZER constraints in Sec. 3.8.

Example Command Sets

(1)
```
TEST.
FOR LIBRARY.
PRINT,DDP.
END FOR.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODLST.
ANALYZER,DDPATHS.
TEST,MODULE = EXPROGM,EXMPL1.
```

APPENDIX A

JOVIAL KEYWORDS RECOGNIZED BY JAVS

The JOVIAL keywords recognized by JAVS are:

ABS	LQ
ALL	LS
AND	MANT
ARRAY	MODE
ASSIGN	MONITOR (2,3)
BEGIN	NENT
BIT	NOT
BYTE	NQ
CHAR	NWDSEN
CLOSE	ODD
COMMON	OPEN
COMPOOL (1)	OR
DEFINE	ORIF
DIRECT	OUT (1)
END	OUTPUT
ENT	OVERLAY
ENTRY	POS
EQ	PROC
FALSE (1)	PROGRAM
FILE	REM
FOR	REMQUO
GOTO	RETURN
GQ	SHUT
GR	START
IF	STOP
IFEITH	STRING
IN (1)	SWITCH
INPUT	TABLE
IO (1)	TERM
ITEM	TEST
JOVIAL	TRUE (1)
LOC	WAIT (1)

NOTES: (1) Honeywell JOVIAL keyword; (2) CDC JOVIAL keyword; (3) JOCIT JOVIAL keyword.

APPENDIX B

ERROR MESSAGES

B.1 UNIVERSAL COMMAND ERROR MESSAGES

Error Message	Corrective Action
Library opened read/write with improper password	Library name for ALTER library must be the same as when created. Change library name and rerun.
No information on read only library	Provide already created library permfile or remove old library command.
Procedure invoked with module equal to zero	Specify a module on the library and rerun.
Too many error messages	Make correction for preceding error messages and rerun.
No JAVSTEXT specified	Specify JAVSTEXT and rerun.

B.2 BASIC ERROR MESSAGES

The error messages* resulting from BASIC source code analysis are:

Error Number	Explanation
0	The first symbol in the JOVIAL source program is not a START or a CLOSE. A START is assumed and processing continues.
1	A single prime which is not a legal symbol in a JOVIAL source program has been encountered. The prime is ignored.
2	The constant has an illegal format. The resulting constant may not compile.
3	A comment is terminated by a dollar sign. The first symbol following the dollar sign is considered to be the beginning of the next statement or declaration.
4	The define information overflows one of the tables provided for it. As a result, this define directive and all subsequent define directives may be ignored.
5	The preset data constant conflicts with the definition of the item with which it is associated. The constant is processed as if it were legal.
6	The symbol encountered in a preset data list or as a single preset data constant is neither a constant nor an END. If the illegal symbol occurs as the first constant in a list or in the place of a single preset data constant, the symbol is changed to the constant zero. If the illegal symbol occurs within a preset data list, it is ignored. If the next symbol in the list is a constant or an END, the processing of the preset data list is continued. If not, the processing is discontinued and the next symbol is considered to be part of the next statement or declaration.
7	An illegal character has been found in the JOVIAL source program. The character is treated as a blank.
8	An end-of-file has been encountered on the input device, but no TERM $ has been found. A TERM $ is assumed and processing continues.
10	The symbol which begins the statement or declaration is not legal at the beginning of a statement or declaration. The JOVIAL source program is scanned until a symbol legal at the beginning of a statement or declaration is encountered.

*BASIC error messages differ from JOVIAL compiler error messages. Many errors can be treated as warnings.

Error Number	Explanation
11	A statement or a declaration which is not legal within a table declaration has been encountered. The processing of the table is terminated. If there are no other declarations for the table, it is removed from the dictionary.* Then the current statement or declaration is processed normally.
12	An E- or A-factor for a constant contains more than the number of numerals allowed for it. The number is truncated on the left.
13	The name is missing from a close, program, or switch declaration. The declaration is ignored.
14	A simple item has been defined by a status constant. An incomplete definition for the item is left in the dictionary.
15	A name other than a sequence designator, simple item, or procedure has been used without being previously declared. This error message is printed only when no COMPOOL has been specified. The name is entered in the dictionary with the standard mode definition, unless it is a table name.
16	There is a syntax error in the switch declaration. The rest of the declaration is ignored.
17	The name appearing in an OPEN or SHUT is not a FILE name.
18	The FILE name does not appear in the proper place in an I/O statement.
19	A constant is succeeded by a loop variable, abbreviation, primitive or name with no intervening blanks. A blank is assumed.
20	A procedure declaration has been encountered within another procedure declaration. The previous procedure is terminated.
21	There is a syntax error in the heading of a procedure declaration. The rest of the symbols up to the first dollar sign are ignored.

*BASIC makes use of an internal dictionary from which JAVS tables are built.

Error Number	Explanation
22	A constant has been found in a subordinate overlay declaration (overlay declaration within a table declaration). The rest of the overlay declaration is ignored.
23	A syntax error has been found in an overlay declaration. The rest of the statement is ignored.
24	The external file name is not a constant or a symbol.
25	A syntax error has been found in a define directive. The rest of the directive is ignored.
26	The name is missing from a simple item, table item, or array declaration. The rest of the declaration is ignored.
27	A STRING statement appears outside a TABLE statement sequence.
28	The dimension information for an array overflows the dimension table. The rest of the array declaration is ignored.
30	Two contiguous symbols in the statement are not legal placed next to one another. The left symbol has been ignored.
31	A possible missing dollar sign has been detected. A dollar sign has been inserted, the statement counter incremented, and the processing of the JOVIAL source program is continued.
32	A symbol has been detected as illegal inside of the current level of bracketers (parentheses, brackets, exponentiation brackets, or absolute value brackets). The statement is processed as though no error were found.
33	A constant followed by an equals sign has been encountered outside of procedure parentheses or switch parentheses. For example: `IF 10 = AA $`
34	A period followed by a right parenthesis has been encountered outside of a 'LOC or procedure parentheses, or a period followed by a comma or an equals sign has been encountered outside of procedure parentheses. If a name preceded the period, it has been treated like a statement name. For example: `AA = AA-(CC+DD.) $`

Error Number	Explanation
35	When the end of the statement was reached, the count of bracketers in the statement had not reached zero. That is, there were more left bracketers than right bracketers in the statement.
36	A right bracketer does not match the appropriate left bracketer. Error messages may be printed for the remaining right bracketers in the statement.
37	Bracketers in the statement are nested more than 24 deep. The processing of the statement has been terminated and the statement counter incremented.
38	A dollar sign has been recognized as the left term in a pair of symbols. This dollar sign is illegal but it is assumed that it terminates the statement. The statement counter is incremented.
40	The accumulator designator "A(" has been encountered outside of a direct statement. It is processed as if there were no error.
41	The entity being declared has the same name as a previously declared entity with overlapping scope and domain. The declaration is processed as though no error has occurred.
42	The entity being declared has the same name as a previously used entity with overlapping scope and domain. The declaration is processed as though no error has occurred.
43	A status constant has been used incorrectly in the statement. This may result in the integer constant 0 being substituted for the status constant.
45	The table containing the names of COMPOOL-defined status variables has overflowed. That is, too many COMPOOL-defined status variables are being used with a total of too many constants. Subsequent usage of new variables or old variables used with new constants will elicit this same error message and will not be handled correctly.
46	The symbol following the FOR in a FOR-clause is not a loop variable. The clause is processed normally.
47	The dictionary has overflowed. That is, the JOVIAL source program contains too many constants and names. Processing of the program has been terminated.

Error Number	Explanation
48	a. A name that is not a sequence designator is being declared, and is found to be the same as a name that is already in the dictionary. or b. A name is being used in a way that is inconsistent with its existing dictionary definition. The name is assumed to be the same as the one in the dictionary.
49	A global simple item has the same name as a procedure. The simple item is processed as though no error has occurred.
50	No dimension information has been given for an array encountered. The preset data are ignored.
51	More than three levels of preset data for an array declaration. The rest of the declaration is processed normally.
52	The preset data list for an array does not have the same number of dimensions as the array. The preset data have been ignored.
53	The count of BEGIN-END's for the preset data for an array or string is incorrect since a symbol illegal within a preset data list has been encountered. The processing of the preset data list is terminated.
54	There is a syntax error in a table declaration. The rest of the declaration is processed in as normal a manner as possible.
55	The number of like tables exceeds 26.
56	A syntax error has been encountered in a simple item, table item, or array declaration. The processing of the declaration is terminated.
57	The status list is missing from a status item declaration. The rest of the declaration is processed normally.
58	A table item declaration does not have the correct format for a defined-entry table. The information in the dictionary for the table item will be incorrect.
59	The overlay information overflows the overlay table. The rest of this overlay declaration is ignored.

Error Number	Explanation
60	A JOVIAL primitive which is not implemented in the JAVS dialect has been encountered. (See Appendix A.) The primitive is ignored.
62	A subsequent constant in a preset data list is not compatible with the initial constant in the list. The constant is processed normally.
64	No dollar sign was found at the end of a statement. A dollar sign was assumed, however.
65	No dollar sign was found at the end of a declaration. A dollar sign was assumed.
66	A TERM or end-of-file on the input device was encountered within a table declaration. Processing of the JOVIAL source program is discontinued.
67	A TERM or end-of-file on the input device was encountered within a procedure. ENDs are added for the procedure. Then the processing of the JOVIAL source program is discontinued.
68	The BEGIN-END count for the main program was not zero when the TERM or end-of-file on the input device was encountered. ENDs are added for the main program. Then the processing of the JOVIAL source program is discontinued.
70	A name or a constant overflows the number of characters allowed for it (120 is the maximum number of characters allowed by most compilers). The excess characters have been deleted from the name or constant.
71	There are too many characters in the names and constants used by the JOVIAL source program. Processing of the JOVIAL source program is discontinued.
73	The number of preset data constants in a list is greater than the maximum number of entries in the table. The excess number of constants has been deleted from the list.
74	A preset data constant (P followed by a constant) was found in an array or table item declaration. The constant was processed normally.
75	An illegal double period has been encountered. A double period is legal only when there are two consecutive floating constants (example: 2..2 which is the same as 2. .2). The second period is processed like a numeral.

Error Number	Explanation
76	A status constant was not found in the status list for the status variable with which it is associated. The value of the constant is set to zero.
77	The statement begins with a dollar sign, and the statement counter is incremented by one.
78	A table declaration with no items has been encountered. The table is removed from the dictionary.
80	A local simple item has the same name as the procedure in which it is being used. This item has no local declaration, but it is assumed that it defines the procedure as a function.
81	A local procedure call references a procedure with the same name as the procedure in which it is being used. This is a recursive use of the procedure name. The two procedure names are assumed to refer to the same entity in the dictionary.
82	A name used in an overlay declaration or as an actual input or output parameters for a procedure has the same name as a table item. The two variables are assumed to have the same entry in the dictionary.
83	A name not identified as to class (i.e., may be a simple item, table item, etc.) has the same name as a procedure with overlapping scope. The variable is processed as though no error has occurred.
84	A table item or table has the same name as a procedure with overlapping scope. The table item or table is processed as though no error has occurred.
85	A local table or array has the same name as the procedure in which it is declared. The table or array is processed as though no error has occurred.
86	A procedure has the same name as another global variable. The procedure name is processed as though no error has occurred.
87	A FILE name matches a PROC name.
88	An impossible entry in the variable or label identification matrix has been encountered. This indicates either a compiler or machine error.

Error Number	Explanation
92	There is a preset data constant for a formal input or output parameter for a procedure. The preset data constant is ignored.
94	There are too many rows in the preset data list for an array. The excess columns have been deleted.
95	There are too many planes in the preset data list for an array. All of the rest of the preset data for the array is ignored.
97	The preset data for a string statement is too long.
98	A BEGIN is missing in preset data.
99	A procedure has been used inconsistently. It has been declared and/or used both as a procedure and a function. The final dictionary definition will reflect the way in which the procedure was declared.
851	More than 3000 names and constants appear in JOVIAL source being processed. Program must be broken down for BASIC analysis.
853	A JAVSTEXT directive does not correspond to text type.
900	A token has been misclassified in BASIC internal processing.
901	More than 53 nested modules appear in JOVIAL source stream. The program must be broken down for BASIC analysis.
941	Module is not a program, procedure, close, COMPOOL, or JAVSTEXT.
950	A CLOSE statement occurs in an illegal context.
984	More than 250 word pairs appear in token. The token is omitted from SB.
985	A COMMON statement not terminated with $. A $ is assumed.
986	An illegal comment in the same form as a JAVS directive has been encountered. The token is not entered in the SB table.
988	More than 100 names appear in the statement. The statement must be broken down for BASIC processing.
990	The statement has an unrecognizable class value.

Error Number	Explanation
997	Input parameter error.
999	An unknown entry type has been encountered in the internal dictionary.

B.3 STRUCTURAL ERROR MESSAGES

Error Message	Corrective Action
Bad statement type on stack	Check syntax of module. Rerun BASIC and STRUCTURAL
BEGIN-END block contains no statement	Informational message only
Duplicate symbol which is not a local label	Duplicate symbol in STB treated as a global label. No rerun necessary
END does not match BEGIN, IFEITH or FOR	Check syntax of module. Rerun BASIC and STRUCTURAL
External array has overflowed	Reduce number of externals to less than 100. Rerun BASIC and STRUCTURAL
External switch call treated as return	Declare switch in module and rerun BASIC and STRUCTURAL
FOR statement not in stack	Rerun STRUCTURAL
FOR target pointer error	Rerun BASIC and STRUCTURAL
FOR variable not found in SB	Check syntax of FOR statement. Rerun BASIC and STRUCTURAL
GOTO label not found	Check syntax of GOTO statement and rerun BASIC and STRUCTURAL on module
Illegal statement type on stack	Rerun STRUCTURAL
Invalid statement in IFEITH/ORIF	Rerun BASIC and STRUCTURAL
Invalid token type in switch declaration	Check syntax of switch declaration and rerun BASIC and STRUCTURAL
JOVIAL statement missing	Check syntax of module for DIRECT-JOVIAL statements. Rerun BASIC and STRUCTURAL
Module number <= 0	Specify a module on the library. Rerun STRUCTURAL

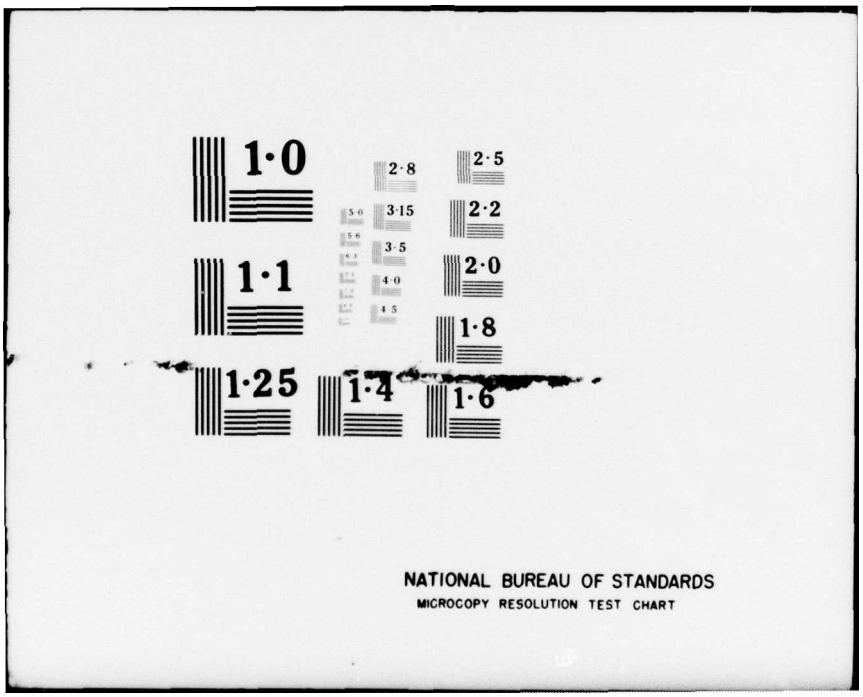
Error Message	Corrective Action
Nested switch call treated as fall through case	Replace switch invocation in switch declaration by a label and rerun BASIC and STRUCTURAL
Never found PROC statement	JOVIAL procedure must contain PROC statement. Rerun BASIC and STRUCTURAL
Pointer LQ 0	Rerun STRUCTURAL
Possible infinite loop detected	Rewrite module to remove infinite loop. Rerun BASIC and STRUCTURAL
Stack overflow	Increase the stack size by recompiling STRUCTURAL. Rerun STRUCTURAL.
Stack underflow	Rerun BASIC and STRUCTURAL
Switch name missing	Check SWITCH syntax and rerun BASIC and STRUCTURAL
Test stack overflow	Increase the size of the test stack by recompiling STRUCTURAL. Rerun STRUCTURAL
Too many paths begin at statement	Increase the size of the DD-path queue by recompiling STRUCTURAL. Rerun STRUCTURAL
Too many statements on DD-path	Increase the size of the DD-path statement list by recompiling STRUCTURAL. Rerun STRUCTURAL
Undefined GOTO	Check location of label referenced in GOTO. Rerun BASIC and STRUCTURAL

AD-A040 104 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JAVS TECHNICAL REPORT. REFERENCE MANUAL.(U)
APR 77 C GANNON, N B BROOKS F30602-76-C-0233
UNCLASSIFIED RADC-TR-77-126-VOL-2 NL

3 OF 3
ADA
040104

END

DATE
FILMED
6-77


1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
5·6
6·3
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

B.4 INSTRUMENT ERROR MESSAGES

Error Message	Corrective Action
Bad module number	Specify a module number within the library. Rerun INSTRUMENT
BEGIN-END block contains no statements	Informative message
Could not match parentheses	Check syntax of statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Delimiter stack overflow	Increase the size of the delimiter stack by recompiling INSTRUMENT. Rerun INSTRUMENT.
Directive not recognized	Check syntax of directive. Rerun BASIC, STRUCTURAL and INSTRUMENT
External switch call not instrumented	Declare switch in module. Rerun BASIC, STRUCTURAL, and INSTRUMENT
GOTO label not found	Check syntax of GOTO statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Input string too long--truncated from head	Change size of internal buffers by recompiling INSTRUMENT. Rerun INSTRUMENT
Invalid statement type in chain	Check syntax of IFEITH statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Invalid switch type	Check syntax of switch declaration. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Last statement in module is not TERM or END	Add a TERM or END statement to the module and rerun BASIC, STRUCTURAL and INSTRUMENT
No DD-path for statement	Rerun STRUCTURAL and INSTRUMENT
Number of switch labels <= 0	Rerun STRUCTURAL and INSTRUMENT
ORIF statement number not on false branch stack	Rerun INSTRUMENT
Pointer out of range	Rerun STRUCTURAL and INSTRUMENT

Error Message	Corrective Action
Stack overflow	Increase the stack size by recompiling INSTRUMENT. Rerun INSTRUMENT
Stack underflow	Rerun INSTRUMENT
String longer than text buffer	Increase the size of the text buffer by recompiling INSTRUMENT. Rerun INSTRUMENT
Unable to recognize type of FOR statement	Check syntax of FOR statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Variable not declared in preceding TRACE directive	Declare variable in TRACE directive. Rerun INSTRUMENT

B.5 ASSIST ERROR MESSAGES

Error Message	Corrective Action
Beginning DDP outside range of DD-paths	Incorrect command. Rerun ASSIST
Target DDP outside range of DD-paths	Incorrect command. Rerun ASSIST
Too many DD-paths	Select smaller reaching set. Rerun ASSIST
Too many DD-paths in reaching set	Select smaller reaching set. Rerun ASSIST
Too many essential predicates	Select smaller reaching set. Rerun ASSIST

B.6 DEPENDENCE ERROR MESSAGES

Error Message	Corrective Action
Too many called modules from group	Select smaller group. Rerun DEPENDENCE
Too many calling modules to group	Select smaller group. Rerun DEPENDENCE
Too many outside modules called from library	Select group off library. Rerun DEPENDENCE

B.7 ANALYZER ERROR MESSAGES

Error Message	Corrective Action
Illegal record on AUDIT file	Check AUDIT file for 4H(TERM) in last record. If not present, test execution did not invoke the test termination PROBI module.
Unknown module specified	Specify only those modules on the library. Rerun ANALYZER.
Bad module value	Check AUDIT file for 4H(TERM) in last record. If not present, test execution did not invoke the test termination PROBI module. Check for use of previously written AUDIT file. Insert test termination PROBI call in instrumented code where it will be invoked during test.

B.8 OTHER JAVS SYSTEM ERRORS*

Error Message

ACTMOD does not know an active module is

Bad call to PUTBIT

Bad module value

Beginning block of search is out of table range

Block does not exist for table

Changed module not yet on new library

Did not know module was active

Entry point tables for both libraries are empty

Fragment beyond range

Fragment number given as zero

GETLST called for non-variable length table

Illegal block number

Illegal core address

Illegal core fragment number

Illegal library address

Illegal library fragment number

Illegal library number

Illegal table number

Improper call to PREPT

Index beyond block size

*The error messages listed in this section are for internal JAVS errors indicative of flaws in JAVS software over which the user has no control.

Error Message
IRSTAB cannot be used with variable length table
Library opened with improper fragment size
LSETUP bad record length
MDBGET can only be used with active modules
MDB not on library
Module location table has been overflowed
Module must be active to change
Module not in core to dump
Module number not yet assigned
Module outside legal range
No more table space available
No such block
No such fragment
Not yet implemented
PUTLST called for non-variable length table
Selected block out of known range
Selected block out of table bounds
Table not known in system
Too many items in variable list

APPENDIX C

STATEMENT TYPE ABBREVIATIONS

Abbreviation	Meaning
ARRY	ARRAY
ASMT	ASSIGNMENT
ASSN	ASSIGN
ASSY	ASSEMBLY CODE
BEGN	BEGIN
CLOS	CLOSE
COMN	COMMON
COMT	COMMENT
CONT	CONTINUATION
CMPL	COMPOOL
DEFN	DEFINE
DIRC	DIRECT
DIRT	JAVS DIRECTIVE
END	END
EXCH	EXCHANGE
FILE	FILE
FOR1	1 FACTOR FOR CLAUSE
FOR2	2 FACTOR FOR CLAUSE
FOR3	3 FACTOR FOR CLAUSE
FUNC	FUNCTION
GOTO	GOTO LABEL
GTCL	GOTO CLOSE
GTSW	GOTO SWITCH
GTEX	GOTO EXTERNAL LABEL (RETURN)
IF	IF
IO	IO
IFEI	IFEITH
IN	IN
INPT	INPUT
INSW	INDEX SWITCH DECLARATION
INV	PROCEDURE INVOCATION
ITEM	ITEM
ITSW	ITEM SWITCH DECLARATION
JOVL	JOVIAL
LABL	LABEL
MODE	MODE
MNTR	MONITOR
OPEN	OPEN
ORIF	ORIF
OUT	OUT
OUTP	OUTPUT
OVRL	OVERLAY
PROB	INSTRUMENTATION PROBE
PROC	PROCEDURE
PROG	PROGRAM
PRST	PRESET LIST
RETN	RETURN
SHUT	SHUT
STRG	STRING
STRT	START
STOP	STOP
SWTC	SWITCH
TABL	TABLE
TBLI	TABLE ITEM
TERM	TERM
TEST	TEST
WAIT	WAIT

INDEX

INDEX (Cont.)

REFERENCES

1. C. Gannon, N. B. Brooks, and R. J. Urban, JAVS Technical Report: Vol. 1, "User's Guide," General Research Corporation, CR-1-722, April 1977.

2. E. F. Miller, Jr., et al., "Structural Techniques of Program Validation," in Proceedings of the IEEE Computer Society International Conference, San Francisco, California, February 1974.

3. JOCIT Compiler User's Manual, Computer Sciences Corporation, August 1974.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

*U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025/220

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.